VLADIMÍR MAŘÍK
OLGA ŠTĚPÁNKOVÁ
JIŘÍ LAŽANSKÝ
a kolektiv

Umělá inteligence (6)

ACADEMIA

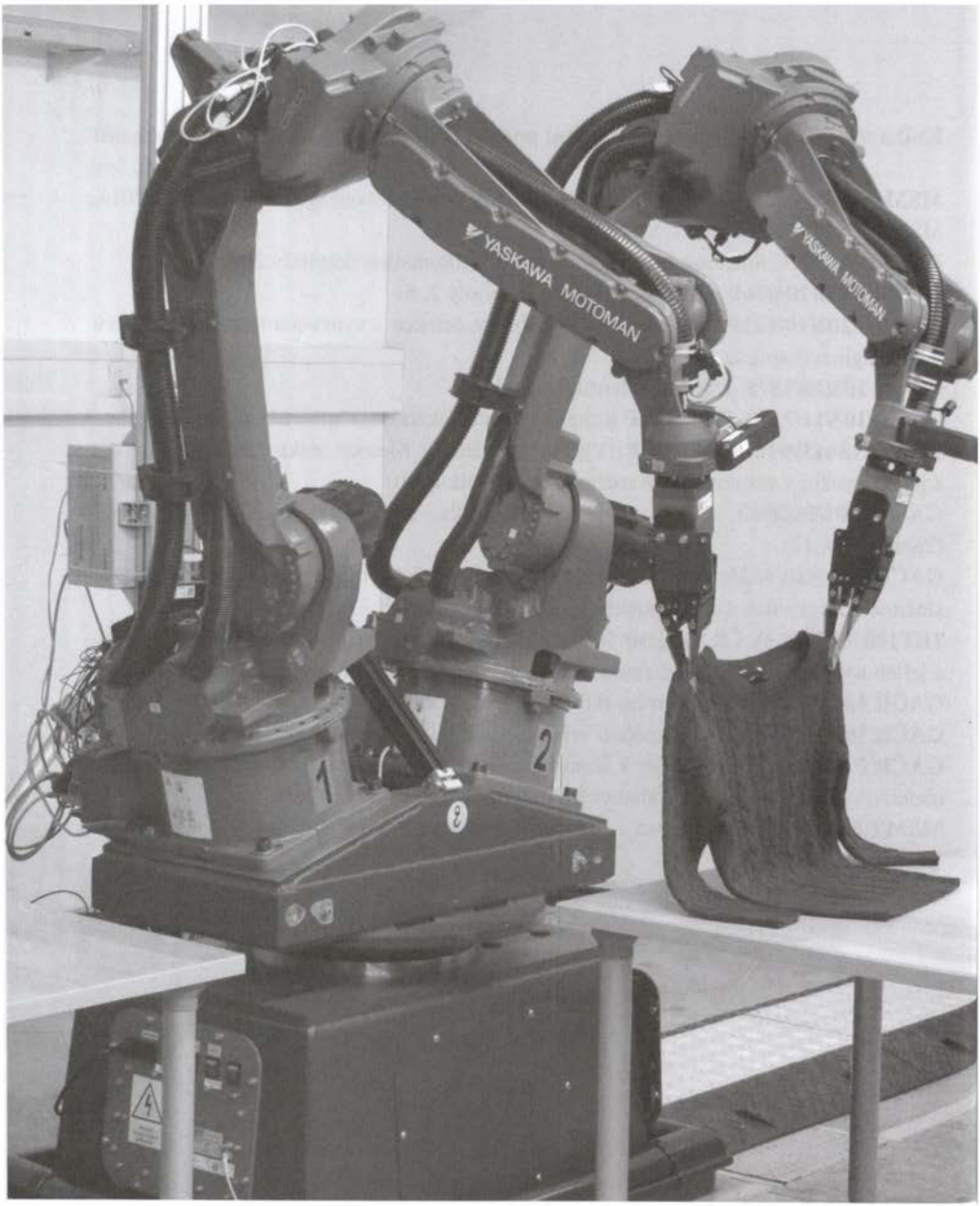
Umělá inteligence (6)

Recenzenti:
prof. Ing. Josef Psutka, CSc.
prof. Ing. Petr Vavřín, DrSc.

Na frontispisu vidíme pohled do experimentálního pracoviště
Katedry kybernetiky ČVUT FEL, představující EU-projekt CLOPEMA,
na němž dvojruký robot právě řeší úlohu skládání oděvu

Kniha vznikla a byla vydána s finanční pomocí těchto grantů a výzkumných úkolů:

MSM 6840770038: Rozhodování a řízení pro průmyslovou výrobu III (2007–2011, MSM) (kapitoly 1, 2, 3, 5, 7, 8)
Projekt RA-DIC financovaný firmou Rockwell Automation (kapitoly 2, 5)
GAČR P202/10/0761: Sémantizace webu (kapitoly 2, 6)
GAČR P202/10/1825: PatOMat – Automatizace detekce a využívání návrhových vzorů v ontologiích (kapitola 4)
GAČR 103/10/1875: Učení se z teorií (kapitola 9)
GAČR 103/11/2170: Převod ILP technik do statistického relačního učení (kapitola 9)
MSM 6138439910 (MŠMT ČR, Výzkumný záměr): Metody získávání znalostí z dat a jejich využití v ekonomickém rozhodování (kapitola 10)
GAČR 201/08/0802: Aplikace metod znalostního inženýrství v dobývání znalostí (kapitoly 10, 11)
GAČR P202/10/1333: NoSCoM: nestandardní výpočetní modely a jejich aplikace ve složitosti, lingvistice a učení (kapitola 12)
1ET100300517 (AV ČR, Program Informační společnost): Metody inteligentních systémů a jejich aplikace při dobývání znalostí a zpracování přirozeného jazyka (kapitola 12)
GAČR 102/08/0593: Modelování BTF textur (kapitola 13)
GAČR 103/11/0335: Přesné měření vzhledu materiálů (kapitola 13)
GAČR P403/12/1557: Přístupy k identifikaci faktorů výkonnosti podniků s důrazem na metody výběru příznaků ve statistickém rozpoznávání (kapitola 13)
MŠMT č. 1M0572: DAR: Data – Algoritmy – Rozhodování (kapitola 13)


YASKAWA MOTOMAN
YASKAWA MOTOMAN
1
2

VLADIMÍR MAŘÍK
OLGA ŠTĚPÁNKOVÁ
JIŘÍ LAŽANSKÝ
a kolektiv

Umělá inteligence (6)

ACADEMIA
PRAHA 2013

KATALOGIZACE V KNIZE – NÁRODNÍ KNIHOVNA ČR

Mařík, Vladimír
Umělá inteligence. (6) / Vladimír Mařík, Olga Štěpánková, Jiří Lažanský a kolektiv. – Vyd. 1. – Praha : Academia, 2013. – 490 s. : il.
ISBN 80-200-0502-1 (soubor). – ISBN 978-80-200-2276-9 (váz.)

004.8 * 004.891:004.41 * 004.62 * 004.8:111.1 * 004.738.52:004.822
- umělá inteligence
- znalostní inženýrství
- zpracování dat
- ontologie (informatika)
- sémantický web
- kolektivní monografie

004.8 – Umělá inteligence [23]

© Vladimír Mařík, Olga Štěpánková, Jiří Lažanský a kol., 2013

ISBN 978-80-200-2276-9
ISBN 80-200-0502-1 (soubor)
ISBN 80-200-0469-3 (1. díl)
ISBN 80-200-0504-8 (2. díl)
ISBN 80-200-0472-6 (3. díl)
ISBN 80-200-1044-0 (4. díl)
ISBN 978-80-200-1470-2 (5. díl)

AUTORSKÝ KOLEKTIV

prof. Ing. Petr Berka, CSc.
berka@vse.cz
VŠE v Praze, Fakulta informatiky a statistiky
Nám. W. Churchilla 4, 130 67 Praha 3

Ing. Jiří Kléma, Ph.D.
klema@labe.felk.cvut.cz
ČVUT v Praze, Fakulta elektrotechnická
Technická 2, 166 27 Praha 6

Ing. Zdeněk Kouba, CSc.
kouba@labe.felk.cvut.cz
ČVUT v Praze, Fakulta elektrotechnická
Technická 2, 166 27 Praha 6

Ing. Petr Křemen, Ph.D.
kremen@labe.felk.cvut.cz
ČVUT v Praze, Fakulta elektrotechnická
Technická 2, 166 27 Praha 6

Ing. Martin Labský, Ph.D.
labsky@seznam.cz
VŠE v Praze, Fakulta informatiky a statistiky
Nám. W. Churchilla 4, 130 67 Praha 3

prof. Ing. Vladimír Mařík, DrSc.
marik@labe.felk.cvut.cz
ČVUT v Praze, Fakulta elektrotechnická
Technická 2, 166 27 Praha 6

doc. RNDr. Jana Novovičová, CSc.
novovic@utia.cas.cz
Akademie věd České republiky
Ústav teorie informace a automatizace
Pod vodárenskou věží 4, 182 08 Praha 8

Ing. Marek Obitko, Ph.D.
marek@obitko.com
Rockwell Automation Research Centre
Pekařská 695/10a, 155 00 Praha 5

prof. Ing. Pavel Pudil, DrSc.
pudil@fm.vse.cz
VŠE v Praze, Fakulta managementu
Jarošovská 1117/II, 377 01 Jindřichův Hradec

prof. RNDr. Jan Rauch, CSc.
rauch@vse.cz
VŠE v Praze, Fakulta informatiky a statistiky
Nám. W. Churchilla 4, 130 67 Praha 3

RNDr. Petr Somol, Ph.D.
somol@utia.cas.cz
Akademie věd České republiky
Ústav teorie informace a automatizace
Pod vodárenskou věží 4, 182 08 Praha 8

doc. Ing. Vojtěch Svátek, Ph.D.
svatek@vse.cz
VŠE v Praze, Fakulta informatiky a statistiky
Nám. W. Churchilla 4, 130 67 Praha

doc. RNDr. Jiří Šíma, DrSc.
sima@cs.cas.cz
Akademie věd České republiky
Ústav informatiky
Pod vodárenskou věží 2, 182 07 Praha 8

prof. RNDr. Petr Štěpánek, DrSc.
petr.stepanek@mff.cuni.cz
Universita Karlova,
Matematicko-fyzikální fakulta
Malostranské náměstí 25, 118 00 Praha 1

prof. RNDr. Olga Štěpánková, CSc.
step@labe.felk.cvut.cz
ČVUT v Praze, Fakulta elektrotechnická
Technická 2, 166 27 Praha 6

RNDr. Josef Urban, Ph.D.
josef.urban@gmail.com
Radboud Universiteit Nijmegen
Heijendaalseweg 135, 6525 AJ Nijmegen, Nizozemsko

Mgr. Ing. Miroslav Vacura, Ph.D.
vacuram@vse.cz
VŠE v Praze, Fakulta informatiky a statistiky
Nám. W. Churchilla 4, 130 67 Praha 3

Ing. Pavel Vrba, Ph.D.
pavel.vrba1@gmail.com
ProTyS, a. s.
Technická 1, 301 00 Plzeň

RNDr. Jiří Vyskočil, Ph.D.
jiri.vyskocil@gmail.com
ČVUT v Praze, Fakulta elektrotechnická
Technická 2, 166 27 Praha 6

Ing. Ondřej Zamazal, Ph.D.
svabo@vse.cz
VŠE v Praze, Fakulta informatiky a statistiky
Nám. W. Churchilla 4, 130 67 Praha

doc. Ing. Zdeněk Zdráhal, CSc.
z.zdrahal@open.ac.uk
The Open University
Human Cognition Research Lab., Milton Keynes MK7 6AA, Velká Británie

doc. Ing. Filip Železný, Ph.D.
zelezny@labe.felk.cvut.cz
ČVUT v Praze, Fakulta elektrotechnická
Technická 2, 166 27 Praha 6

Obsah

Předmluva ... 19

1 Ontologie: od filosofie k umělé inteligenci ... 21
Z. Zdráhal

1.1 Filosofická východiska, základní pojmy ... 21
1.2 Základní pojmy a definice ... 24
1.3 Konceptualizace ... 25
1.4 Jednoduchý model komunikace ... 26
1.5 Glosář a tezaurus ... 28
1.6 Ontologie ... 32
1.6.1 Jednoduchý příklad ... 34
1.7 Klasifikace ontologií podle principů modelování ... 36
1.8 V praxi využívané kategorie ontologií a vztahy mezi nimi ... 38
1.9 Mapování ontologií, základní ontologie a sémantické lexikony ... 40
1.9.1 WordNet ... 45
1.10 Metodiky pro návrh ontologií ... 46
1.10.1 Návrh ontologie podle L. Noyové a D. McGuinnessové ... 47
1.10.2 Metodika Methontology ... 50
1.10.2.1 Životní cyklus ... 50
1.10.2.2 Konceptualizace ... 52
1.10.3 Metodika DILIGENT ... 54
1.11 Jazyk, popis světů a reprezentace znalostí ... 56
1.11.1 Nutnost, možnost a metavlastnosti ... 58
1.12 Metodika pro analýzu ontologií – OntoClean ... 59
1.12.1 Rigidnost ... 59
1.12.2 Identita ... 60
1.12.3 Jednota ... 62
1.12.4 Příklad použití metodiky OntoClean ... 63
1.12.4.1 Krok 1. Posouzení metavlastností jednotlivých tříd ... 63
1.12.4.2 Krok 2. Rigidní třídy – vytvoření páteřní taxonomie ... 65
1.12.4.3 Krok 3. Ověření kritéria identity ... 65
1.12.4.4 Krok 4. Ověření kritéria jednoty ... 66
1.12.4.5 Krok 5. Ověření kritéria jednoty a antijednoty ... 67
1.12.4.6 Krok 6. Začlenění nerigidních tříd ... 67
1.13 Příklad: Ontologie CIDOC CRM ... 68
1.14 Formální reprezentace ontologií ... 71
1.14.1 Rámcové modely ... 73
1.14.2 Sémantické sítě ... 73
1.14.3 Konceptuální grafy ... 74
1.14.4 Knowledge Interchange Format (KIF) ... 75
1.14.5 Ontolingua ... 75

1.14.6 Deskripční logiky 77
1.14.7 Jazyky sémantického webu (RDF, RDFS, OWL) 78
1.15 Příklady aplikací 79
1.16 Závěr 81
Literatura 81

2 Ontologie a sémantický web 85
M. Obitko, O. Zamazal, V. Svátek

2.1 Sémantický web 85
2.2 Reprezentace znalostí na sémantickém webu 86
2.2.1 Architektura sémantického webu 86
2.2.2 RDF – Resource Description Framework 88
2.2.3 RDFS – RDF Schema 92
2.2.4 OWL – Web Ontology Language 95
2.2.4.1 Třídy a individua 98
2.2.4.2 Objektové a datové vlastnosti 99
2.2.4.3 Modelování tříd 101
2.2.4.4 Vlastnosti pro anotování 102
2.2.4.5 Importování ontologií a předpoklad otevřeného světa 103
2.2.4.6 Úrovně a profily jazyka OWL 104
2.2.5 Dotazovací jazyk SPARQL 105
2.2.6 Spojení s dokumenty na webu 106
2.3 Zpracování znalostí na sémantickém webu 107
2.3.1 Editory ontologií v jazyce OWL 108
2.3.1.1 Protégé 108
2.3.1.2 Ostatní editory 109
2.3.2 Strojové odvozování 111
2.3.3 Vyhledávání a mapování ontologií 112
2.3.3.1 Metody a nástroje pro vyhledávání ontologií 112
2.3.3.2 Mapování a integrace ontologií 113
2.3.3.3 Aplikace mapování ontologií 114
2.3.3.4 Metody mapování 114
2.3.3.5 Příklady systémů 115
2.3.3.6 Vyhodnocování výsledků mapování ontologií 115
2.3.3.7 Současné trendy v mapování ontologií 116
2.4 Praktické aplikace sémantického webu 117
2.4.1 Sociální sítě 117
2.4.2 Hesláře a tezaury 117
2.4.3 Kolaborativní publikování 118
2.4.4 Vývoj softwaru 118
2.4.5 Elektronické obchodování 118
2.4.6 Linked data 120
2.5 Závěr 121
Literatura 123

3 Ontologie a deskripční logiky 126
P. Křemen

3.1 Úvod 126
3.2 Deskripční logika *ALC* 126
3.2.1 Syntax a sémantika jazyka *ALC* 127
3.2.2 Inference, metody automatického dokazování 129
3.2.3 Tablové algoritmy 131
3.2.3.1 Grafy zúplnění 131
3.2.3.2 Tablový algoritmus pro jazyk *ALC* při prázdném TBoxu 131
3.2.3.3 Tablový algoritmus pro jazyk *ALC* v obecném případu 136
3.3 Expresivní deskripční logiky 138
3.4 Pokročilé inferenční procedury 140
3.4.1 Vysvětlování modelovacích chyb 140
3.4.1.1 Metody glass-box 141
3.4.1.2 Metody black-box 141
3.4.2 Dotazování v deskripčních logikách 144
3.4.2.1 Konjuktivní ABox dotazy 145
3.5 Deskripční logiky a pravidlové systémy 147
3.6 Závěrečné poznámky 147
Literatura 148

4 Ontologické inženýrství na sémantickém webu 149
V. Svátek, M. Vacura

4.1 Úvod 149
4.2 „Vnitřní rozměr" ontologického inženýrství – struktura a sémantika jazyka .. 150
4.2.1 Volba typů entit v jazyce OWL 150
4.2.2 Jmenné konvence pro názvy entit 152
4.2.3 Elementární struktury a inferenční efekty 154
4.2.4 Rozsáhlejší logicko-strukturní návrhové vzory 156
4.3 „Vnější rozměr" ontologického inženýrství – přepoužití externích zdrojů 158
4.3.1 Jednoduché ontologie a slovníky 158
4.3.2 Obsahové návrhové vzory 158
4.4 Využívání ontologických vzorů jako systematická disciplína 160
4.4.1 Klasifikace vzorů 160
4.4.2 Příklad implementovaného vzoru 162
4.4.3 Metodika eXtreme Design 164
4.5 Diskuse a shrnutí 165
Literatura 167

5 Aplikace ontologií v průmyslové výrobě 169
P. Vrba, M. Obitko, V. Mařík

5.1 Úvod 169
5.2 Hledání ve strukturovaných datech 170
5.2.1 Popis domény 171

5.2.1.1 Řídicí kód pro programovatelné automaty ... 171
5.2.1.2 Rozhraní člověk–stroj ... 172
5.2.2 Vyhledávání ... 173
5.2.2.1 Architektura systému RoSS ... 174
5.2.2.2 Převod dat do formátu RDF/OWL ... 175
5.2.2.3 Hledání pomocí jazyka SPARQL ... 175
5.2.2.4 Uživatelské rozhraní pro konstrukci dotazů ... 177
5.2.2.5 Shrnutí ... 180
5.3 Sémantika v holonických a multiagentních průmyslových řídicích systémech ... 181
5.3.1 Agentové technologie pro automatizaci ... 181
5.3.2 Doménově specifické ontologie ... 183
5.3.3 Obecné ontologie ... 185
5.4 Návrh obecné ontologie a multiagentní architektury pro automatizaci ... 186
5.4.1 Ontologie ... 186
5.4.2 Integrace ontologie v distribuovaném řídicím systému ... 188
5.5 Sémantické rozšíření agentního řídicího systému přepravy materiálu ... 190
5.5.1 Systém MAST ... 190
5.5.2 Případová studie – balicí linka na Univerzitě v Cambridgi ... 192
5.6 Závěr ... 194
Literatura ... 196

6 Automatické sémantické anotování a extrakce informací ... 200
M. Labský, V. Svátek

6.1 Využití extrakce informací a sémantického anotování ... 200
6.1.1 Úvod ... 200
6.1.2 Anotování a sémantika ... 201
6.1.3 Stručná typologie anotačních přístupů ... 202
6.1.3.1 Co je anotováno ... 202
6.1.3.2 Kdo anotuje ... 203
6.1.3.3 Čím se anotuje ... 203
6.1.3.4 O čem anotace vypovídá ... 204
6.1.3.5 Kde je anotace uložena ... 204
6.2 Scénáře pro ruční anotování ... 205
6.2.1 Sémantická tvorba dokumentů ... 205
6.2.2 Sekundární ruční anotace dokumentů ... 206
6.2.3 Ruční anotace trénovacích dat ... 207
6.3 Automatická extrakce informací: typy úloh a evaluace ... 208
6.3.1 Motivace ... 208
6.3.2 Kritéria pro členění úloh extrakce informací ... 209
6.3.3 Měření úspěšnosti extrakce informací ... 211
6.4 Přehled metod pro automatickou extrakci informací ... 213
6.4.1 Metody založené na ručně zadaných pravidlech ... 213
6.4.1.1 Extrakce informací založená na ontologiích ... 213

6.4.1.2 Jazyky pro tvorbu extrakčních pravidel ... 214
6.4.2 Algoritmy strojového učení nevyžadující pravidelné formátování ... 214
6.4.2.1 Algoritmy dle reprezentace naučených znalostí a principu fungování ... 214
6.4.2.2 Algoritmy dle typu zpracovávaných dat ... 215
6.4.2.3 Možné reprezentace dokumentů ... 216
6.4.3 Wrappery ... 217
6.4.3.1 Ruční tvorba wrapperů ... 218
6.4.3.2 Supervizovaně trénované wrappery ... 218
6.4.3.3 Nesupervizovaně trénované wrappery ... 219
6.4.4 Aktivní učení ... 219
6.4.5 Bootstrapping ... 220
6.5 Vybrané metody pro extrakci informací ... 221
6.5.1 Extrakční ontologie ... 221
6.5.2 Transformační pravidla pro extrakci informací ... 225
6.5.2.1 Agoritmus LP^2 ... 225
6.5.2.2 Sémantické parsování pomocí transformačních pravidel ... 226
6.5.3 Grafické pravděpodobnostní modely ... 227
6.5.3.1 Skryté Markovovy modely ... 229
6.5.3.2 Markovovy modely s maximální entropií ... 236
6.5.3.3 Podmíněná náhodná pole – Conditional Random Fields ... 238
6.5.3.4 Jiné grafické modely ... 240
6.6 Využití extrakčních metod pro sémantizaci webu ... 240
6.7 Závěrečné poznámky ... 242
Literatura ... 244

7 Automatické uvažování ... 249
J. Urban, J. Vyskočil, P. Štěpánek

7.1 Co je automatické dokazování vět ... 249
7.2 K čemu je automatické dokazování vět ... 252
7.2.1 Použití v rámci interaktivních dokazovačů ... 252
7.2.2 Tvorba a verifikace softwaru ... 252
7.2.3 Verifikace hardwaru ... 253
7.3 Potřebné pojmy z matematické logiky ... 254
7.4 Rezoluční automatické dokazování ... 256
7.4.1 Rezoluční metoda ... 256
7.4.2 Smyčka ANL ... 257
7.4.3 Faktorizace a další pravidla ... 261
7.4.4 Jazyk TPTP ... 262
7.4.5 Automatické řešení problému dokazovačem ... 264
7.4.6 Interpretace nalezeného automatického řešení problému ... 269
7.5 Metoda tableau ... 269
7.5.1 Pravidla tableau ... 270
7.5.2 Lean TAP ... 273
7.6 Kam dále? ... 277
Literatura ... 277

8 Modální a temporální logiky 279
O. Štěpánková, P. Štěpánek

8.1 Úvod do modální logiky 280
8.1.1 Sémantika modální logiky 281
8.1.2 Validní formule modální logiky 289
8.1.3 Axiomatický systém a jeho vlastnosti 290
8.1.4 Společná znalost (Common Knowledge) 294
8.2 Temporální logiky. 296
8.2.1 Lineární temporální logika 297
8.2.2 Sémantika pro Lineární Temporální Logiku (LTL) 298
8.2.3 Dokazatelnost v LTL 302
8.3 Závěrečné poznámky 306
Literatura 307

9 Využívání znalostí pro získávání znalostí 308
F. Železný, J. Kléma

9.1 Zaujetí algoritmu 309
9.2 Parametrické metody 312
9.2.1 Grafové modely 314
9.2.2 Logické modely 316
9.2.3 Bayesovský přístup 316
9.3 Neparametrické metody 317
9.3.1 Syntax vzorů 318
9.3.2 Explicitní vyjádření apriorní znalosti 319
9.3.3 Volné zaujetí 320
9.4 Metaznalosti 320
9.5 Závěrečná poznámka 322
Literatura 322

10 Adaptivní přístupy k dobývání znalostí 325
P. Berka

10.1 Úvod 325
10.2 Strojové učení a dobývání znalostí 326
10.3 Vybrané adaptivní přístupy 330
10.3.1 Inkrementální učení 331
10.3.1.1 Rozhodovací stromy 331
10.3.1.2 Rozhodovací pravidla 332
10.3.1.3 Neuronové sítě 334
10.3.1.4 SVM 335
10.3.1.5 Učení založené na instancích 335
10.3.2 Integrování znalostí 336
10.3.3 Revize znalostí 337
10.3.3.1 Induktivní logické programování 337
10.3.3.2 Revize pravidel 338

10.3.4 Posun konceptu ... 340
10.3.4.1 Učení a zapomínáni ... 341
10.3.4.2 Koncepty závislé na kontextu ... 344
10.3.5 Analogie a adaptace ... 344
10.4 Závěr ... 345
Literatura ... 346

11 Metoda GUHA a dobývání znalostí z databází ... 348
J. Rauch

11.1 Úvod ... 348
11.2 GUHA a asociační pravidla ... 349
11.2.1 Procedura ASSOC ... 350
11.2.2 Příklady aplikací procedury 4ft-Miner ... 352
11.2.2.1 Analytické otázky ... 356
11.2.2.2 Booleovské charakteristiky skupin atributů ... 358
11.2.2.3 Bool(Osobní charakteristiky, Tělesné aktivity) $\Rightarrow_{p,Base}$ Bool(Měření) ... 360
112.2.4 Bool(Osobní charakteristiky, Tělesné aktivity) $\Rightarrow^{+}_{p,Base}$ Bool(Měření) ... 363
11.2.2.5 Bool(Osobní charakteristiky, Tělesné aktivity) $\Rightarrow^{+}_{p,Base}$ Bool(Rizika) ... 364
11.2.3 ASSOC a neúplná informace ... 365
11.2.4 ASSOC a analýza nákupního košíku ... 369
11.3 Systém LISp-Miner ... 371
11.3.1 GUHA Procedura SD4ft-Miner ... 372
11.3.2 Další GUHA procedury systému LISp-Miner ... 374
11.4 Observační kalkuly ... 377
11.4.1 Matematické základy automatické tvorby hypotéz ... 377
11.4.2 Observační kalkuly asociačních pravidel ... 378
11.4.3 Třídy 4ft-kvantifikátorů ... 380
11.4.4 Dedukční pravidla pro asociační pravidla ... 382
11.5 Související výzkum ... 383
11.5.1 Výzkumná témata ... 384
11.5.2 Projekt SEWEBAR ... 387
Literatura ... 389

12 Taxonomie výpočetních modelů neuronových sítí: od subregulárních jazyků k superturingovským výpočtům ... 392
J. Šíma

12.1 Úvod ... 392
12.1.1 Výpočetní teorie neuronových sítí ... 392
12.1.2 Taxonomie formálních výpočetních modelů neuronových sítí ... 393
12.1.3 Perceptronové sítě ... 395
12.2 Jednotlivý perceptron ... 396

12.3 Dopředné sítě ... 397
12.3.1 Binární dopředné sítě ... 397
12.3.1.1 Výpočetní univerzalita ... 398
12.3.1.2 Polynomiální váhy ... 398
12.3.1.3 Omezený vstupní stupeň hradel ... 398
12.3.1.4 Polynomiální velikost a konstantní hloubka ... 399
12.3.1.5 Symetrické booleovské funkce ... 400
12.3.1.6 Aritmetické funkce ... 401
12.3.1.7 Celková délka propojení ... 401
12.3.2 Analogové dopředné sítě ... 402
12.3.2.1 Konstantní velikost ... 402
12.3.2.2 Polynomiální velikost ... 403
12.4 Rekurentní neuronové sítě ... 403
12.4.1 Konečné neuronové akceptory jazyků ... 404
12.4.2 Konečné asymetrické rekurentní sítě ... 405
12.4.2.1 Binárn í sítě ... 405
12.4.2.2 Analogové sítě s racionálními váhami ... 406
12.4.2.3 Analogové sítě s reálnými váhami ... 406
12.4.2.4 Analogový šum ... 407
12.4.2.5 Problém zastavení ... 407
12.4.3 Konečné symetrické rekurentní sítě ... 407
12.4.3.1 Konvergence ... 407
12.4.3.2 Čas konvergence ... 408
12.4.3.3 Stabilní stavy ... 409
12.4.3.4 Problém minimální energie ... 409
12.4.3.5 Výpočetní síla ... 409
12.4.4 Nekonečné posloupnosti binárních rekurentních sítí ... 410
12.5 Pravděpodobnostní neuronové sítě ... 411
12.5.1 Pravděpodobnostní dopředné sítě ... 411
12.5.2 Pravděpodobnostní rekurentní sítě ... 412
12.6 Spojitý čas ... 413
12.7 Závěr ... 413
Příloha ... 414
Literatura ... 416

13 Moderní metody výběru příznaků ve statistickém rozpoznávání ... 424
P. Somol, J. Novovičová, P. Pudil

13.1 Úvod 424
13.2 Redukce dimenzionality ... 424
13.2.1 Redukce dimenzionality podle charakteru výsledných příznaků ... 425
13.2.2 Redukce dimenzionality podle cíle ... 425
13.3 Výběr podmnožiny příznaků ... 426
13.3.1 Volba metod výběru příznaků podle optimality ... 426
13.3.2 Volba metod výběru příznaků podle způsobu vyhodnocování kritéria ... 427

13.3.3 Volba metod výběru příznaků podle znalosti problému ... 427
13.4 Optimální vyhledávací metody ... 428
13.4.1 Základní pojmy a varianty algoritmu větví a mezí ... 429
13.4.2 Nevýhody tradičních algoritmů větví a mezí ... 430
13.4.3 Vylepšení „vylepšeného" algoritmu ... 431
13.4.4 Rychlý algoritmus větví a mezí ... 434
13.4.5 Nové vlastnosti algoritmů využívajících predikce ... 438
13.4.5.1 Specifické vlastnosti rychlého algoritmu větví a mezí ... 438
13.4.6 Experimenty s optimálním výběrem příznaků ... 439
13.4.6.1 Syntetické experimenty ... 439
13.4.6.2 Experimenty na reálných datech ... 441
13.4.7 Shrnutí optimálních metod ... 443
13.5 Suboptimální vyhledávací metody ... 444
13.5.1 Individuálně nejlepší příznaky ... 445
13.5.2 Sekvenční vyhledávání ... 445
13.5.3 Nejjednodušší sekvenční výběr ... 446
13.5.4 Sekvenční plovoucí vyhledávání ... 447
13.5.5 Další rozvoj principu plovoucího vyhledávání ... 448
13.5.6 Oscilační vyhledávání ... 448
13.5.7 Experimentální porovnání *d*-parametrizovaných metod ... 451
13.6 Optimalizace velikosti podmnožiny – dynamické oscilační vyhledávání ... 453
13.6.1 Experimentální porovnání *d*-optimalizujících metod ... 454
13.7 Hybridní algoritmy ... 455
13.8 Výběr příznaků založený na modelu směsi hustot ... 457
13.8.1 Konečná směs hustot pravděpodobnosti ... 458
13.8.2 Modifikovaná konečná směs součinových komponent ... 458
13.8.3 Míry významnosti příznaků při užití směsového modelu ... 460
13.8.4 Odvození rozhodovacího pravidla a souhrn vlastností směsového přístupu ... 460
13.8.5 Experiment na reálných datech ... 461
13.9 Problém přeučení a problém stability výběru příznaků ... 461
13.9.1 Problém stability výběru příznaků ... 462
13.9.2 Vybrané míry stability výběru příznaků ... 462
13.9.3 Experimenty s mírami stability ... 465
13.10 Shrnutí a další vývoj ... 466
Literatura ... 468

Česko–anglický slovník ... 472

Anglicko–český slovník ... 479

Rejstřík ... 486

Předmluva

Šestý díl monografie, který právě dostáváte do rukou, organicky navazuje na pět předchozích dílů, které nakladatelství AV ČR, Academia, postupně vydávalo od roku 1993 a jež sehrály svoji roli při vytváření české terminologie, při výchově doktorandů a studentů magisterského studia na vysokých školách jak v České republice, tak i na Slovensku.

Zatímco první dva díly poskytují trvale platné, svým způsobem „stabilní" základy umělé inteligence jako vědní disciplíny, další díly jdou do větších detailů a hloubky. Jsou speciálnější a podávají průběžný přehled rozvoje vědní disciplíny a jejích technik, které vždy vycházejí jak z možností a výkonnosti výpočetních technologií, tak i z potřeb praxe. Jsme přesvědčeni, že ani tyto další díly knihy nestárnou a stále poskytují čtenářům kvalitní zdroj informací.

Šestý díl se věnuje především filosofickým základům, novým teoretickým znalostem a současnému stavu nejnovějších technik umělé inteligence, které se přirozeně vyvíjejí v souvislosti s rozvojem výpočetních prostředků, distribuovaných systémů a celosvětového webu. V důsledku toho se většina kapitol zabývá rozvojem možností pro reprezentaci znalostí, zejména znalostí ontologických a sémantických jako znalostí vyššího typu, moderními netriviálními logikami jako nástroji reprezentace znalostí a uvažování, ale i dobýváním znalostí z rozsáhlých souborů dat.

Jak je již typické pro naši řadu publikací, začínáme od filosofických otázek a přes stále techničtěji orientované kapitoly docházíme až ke kapitolám s vyloženě aplikační tematikou, v našem případě až k popisu technických systémů agentně orientovaného automatického řízení, u něhož hraje významnou roli právě sémantická informace.

Zvláštní pozornost je věnována též výpočetním možnostem neuronových sítí i výběru příznaků v oblasti statistického rozhodování. I když se ladění těchto kapitol může na první letmý pohled zdát nesourodé se zbylou větší částí knihy, dospěli jsme po zralé úvaze k názoru, že právě ony nabízejí velmi cenný příspěvek do celkového obrazu nástrojů pro reprezentaci znalostí. Tyto systémy používají alternativní přístup, který prokázal své výhody v některých reálných aplikacích umělé inteligence, a tedy systémově do knihy dobře zapadají. Snaží se totiž o efektivní reprezentaci a redukci složitosti v jasně vymezených teoretických rámcích neuronových sítí či statistických rozhodovacích metod. Publikace tak ve svém celku může českému čtenáři poskytnout velmi plastickou mozaiku toho, co se u nás na vysoké mezinárodní úrovni v oblasti moderní reprezentace znalostí zkoumá a jak je celá tato „první linie" výzkumu nejen u nás, ale i ve světě rozmanitá ve svých přístupech a výsledcích.

Věříme, že se „Umělá inteligence 6" bude čtenářům líbit, že ocení naše úsilí o vyváženost přehledovosti a hloubky informací na jedné straně, ale i vyváženost teorie a aplikací na straně druhé. Šlo nám též o to, aby autorský kolektiv tvořili co nejšpičkovější čeští odborníci v předmětných tématech a aby jejich kapitoly do sebe tematicky a terminologicky co nejlépe zapadaly. Jsme rádi, že všichni, které jsme oslovili, a to bez výjimky, se své role ujali a věnovali svým kapitolám mimořádné úsilí s cílem pomoci vytvořit velmi kvalitní, pro českého čtenáře tolik potřebnou publikaci. Všechny kapitoly byly ze strany editorů pečlivě čteny a ve většině případů bylo potřeba několika iterací k dosažení patřičné míry konzistence s ostatními. Při vysokém pracovním vytížení všech autorů jim za trpělivost s editory, vstřícnost při řešení nekonzistencí a úsilí o nejvyšší kvalitu patří náš vřelý dík a mimořádný obdiv!

Věřte nám, že sestavit publikaci, která by byla opravdovým přínosem pro celé široké spektrum odborné veřejnosti, zejména však doktorandům a studentům mnoha nejrůznějších oborů, nebylo úkolem lehkým. Sami posuďte, jak se nám náš záměr podařil a do jaké míry je kniha užitečnou.

Vladimír Mařík
Olga Štěpánková
Jiří Lažanský

1 Ontologie: od filosofie k umělé inteligenci

1.1 Filosofická východiska, základní pojmy

Slovo ontologie bylo převzato z filosofie, kde označuje disciplínu, která se zabývá nejobecnějšími otázkami týkajícími se jsoucna a bytí. Toto terminologické převzetí není náhodné, ale bylo vedeno vědomím věcné souvislosti mezi předmětem určitého typu výzkumu v informatice se zkoumáními, která tradičně spadají do této filosofické disciplíny.

Jednou ze základních součástí většiny informačních systémů je explicitní či implicitní popis (model, zobrazení) části reality, ke které se informační systém určitým způsobem vztahuje. V případě prvních jednoduchých informačních systémů byla tato část reality dobře vymezena a předmětem zpracování byly pouze informace o omezené množině předem dobře známých typů objektů. S tím, jak informační systémy postupně zahrnovaly informace o stále rozsáhlejších a komplexnějších částech reality, byly zároveň vyvíjeny metody a nástroje podporující explicitní artikulaci popisů a modelů, které stály u jejich zrodu. S příchodem webu se situace ještě zkomplikovala, protože, ačkoliv se na webu můžeme setkat pouze s formálně omezeným počtem typů informačních objektů, ze sémantického hlediska se informace, které zde nalezneme, mohou týkat zcela čehokoliv. Nevystačíme tedy s těmi metodami, které předpokládají zachycení pouze určitého omezeného výseku reality, ale bylo nezbytné vypracovat takové přístupy, které počítají s neomezenou sémantikou těchto informačních objektů. Podobně se v případě snahy o vzájemnou interakci informačních systémů zpracovávajících podstatně sémanticky odlišné typy informací ukázala potřeba nezávislého referenčního rámce, v němž by bylo možné explicitně popsat jejich vzájemný vztah.

V těchto případech se pak projevila úzká souvislost mezi výzkumem informačních a znalostních technologií a filosofickými ontologickými teoriemi. Jak uvidíme v dalších kapitolách, jedná se zejména o oblasti základních ontologií a obecné metodiky tvorby ontologií (viz oddíl 1.12 o metodice OntoClean). To je také důvodem, proč tento text zahrnuje stručný úvod věnovaný historickým souvislostem s filosofickou ontologií.

Mezi první ontologické koncepce patřily už výklady světa předložené předsokratovskými mysliteli [1], které se povětšinou omezily na snahu redukovat rozličnost smyslového světa na určitý základní princip (arché). Za zmínku zde stojí pýthagorejská škola,

[1] Za zakladatele filosofické ontologie je tradičně považován Parmenidés a za první dílo věnující se ontologii jeho báseň *O přírodě*.

která přichází s prvním vymezením určitých základních, dále neredukovatelných kategorií.[2] Další pokrok přineslo až myšlení Platóna (Reale, 2005) a posléze jeho žáka Aristotela, u něhož se zastavme, protože u něj nacházíme první ontologickou koncepci, kterou můžeme považovat za předchůdce moderních ontologií. Aristotelés svoji teorii předkládá ve spisech *Metafyzika* a *Kategorie*, kde mluví o tzv. první vědě, která kromě ostatních určení zkoumá zejména „jsoucí jakožto jsoucí a to, co mu samému náleží". Výrazy „metafyzika", „ontologie" a aristotelská „první filosofie" se často ve filosofii používají jako synonyma.

Zatímco se speciální vědy věnují pouze určitému vymezenému okruhu jsoucen nebo určitým vymezeným aspektům skutečnosti (například biologie se věnuje jen tomu, co je živé), filosofická ontologie se v tomto pojetí věnuje charakteristikám, které jsou relevantní vzhledem ke všemu jsoucímu. Aristotelés se však už nesnaží o pouhou redukci veškeré skutečnosti na jeden základní princip, jak to činili předsokratovští myslitelé, ale snaží se rozpoznat základní vlastnosti a rozčlenění veškeré skutečnosti. Základními protikladnými charakteristikami, které prostupují napříč všemi speciálními oblastmi vědeckého zkoumání, pak jsou: akt a potence, substance a akcident, látka a forma. Pro nás je podstatné rozlišení, které předkládá ve spisu *Kategorie*, kde nejprve na základě dvou kritérií zavádí rozčlenění všeho jsoucího na čtyři základní třídy: první a druhou podstatu, obecné a jednotlivé akcidenty. Následně pak rozlišuje tzv. deset kategorií: podstatu (substanci)[3] a devět akcidentů: kvantitu, kvalitu, relaci, kde, kdy, polohu, mít, činnost a trpnost. Těchto deset kategorií mělo vyčerpat všechny základní třídy toho, co může v nějakém smyslu být jsoucím. Nezávisle existují vždy pouze jsoucna kategorie substance, zatímco jsoucna ostatních kategorií vždy na bytí nějaké substance participují. Například červená, jakožto určitá kvalita, nemá samostatné bytí, ale vždy existuje jen prostřednictvím nějaké červené věci. Přitom Aristotelés rozlišuje červenou jakožto obecnost (daný odstín) a jako jednotlivost, tedy vyskytující se na určitém konkrétním objektu. Řada současných informatických základních ontologií (viz oddíl 1.8) vychází z rozlišení vypracovaných už Aristotelem. Pro nás je také zajímavé, že kategorie (zejména substance), podobně jako moderní informační ontologie, připouštějí vnitřní hierarchickou strukturu. Není přitom jasné, nakolik sám Aristotelés tuto hierarchickou strukturu například v případě přírodovědných oborů, jako je zoologie či botanika, rozpracoval, protože ačkoli víme, že se této problematice věnoval, řada těchto spisů je ztracena (Vacura, 2009).

Nejstarším dochovaným rozpracováním této hierarchické struktury se tak stává spis novoplatonika Porfyria (234–305) nazvaný *Isagoge*, v podstatě úvod do Aristotelova spisu *Kategorie*. Toto dílo obsahuje tzv. Porfyriův strom (*Arbor porphyriana*), který je hierarchickou klasifikací aristotelských substancí, ze které dodnes vychází hierarchický klasifikační systém používaný v biologii (Hennig, 1950).

---

[2] Jednalo se o následující protikladné dvojice: omezené a neomezené, sudé a liché, jedno a množství, pravé a levé, mužské a ženské, věc v klidu a věc v pohybu, přímé a zakřivené, světlo a tma, dobré a zlé, čtverec a obdélník.

[3] Pro Aristotela je ve spisu *Kategorie* základní (první) substancí konkrétní objekt (např. Sókratés). Následně zavádí tzv. druhé substance, což jsou obecnosti (např. člověk). Substance (řec. ousia, lat. *substantia* – to, co stojí pod) tak představuje jakousi základní jednotku skutečnosti, to skutečně jsoucí, proto u různých filosofů nabývalo toto slovo různých významů (např. pro atomisty jsou atomy substancemi).

Alternativní klasifikační systém se objevil až ve 13. století, kdy se Ramon Llull (1232–1316) ve svém hlavním díle *Ars Magna* pokusil vypracovat logicko-kombinatorický grafický systém pro klasifikaci veškerého poznání, nezávislý na jazyce, náboženství a kultuře. Toto dílo mělo vliv na rozvoj matematické kombinatoriky například u G. W. Leibnize, ovšem jakožto klasifikační systém se neprosadilo. Ve středověku se pak rozvíjením klasifikačních přístupů, především v Porfyriově tradici, zabýval třeba W. Ockham (1280–1349) a další.

Samotný výraz „ontologie" je poměrně nový, objevil se v 17. století v pracích německých filosofů R. Göckela (1613) a J. Lorharda (1606) a prosazený byl patrně Ch. Wolffem, jedním z nejvýznamnějších německých filosofů před I. Kantem, který ho použil pro nahrazení tradičního termínu „metafyzika" ve smyslu nauky o jsoucnu a bytí jako takovém ve spise *Philosophia prima sive ontologia* z r. 1736 (Øhrstrøm, 2005), (Smith, 2004). Wolff byl systematickým filosofem, který se ve svém díle snažil integrovat aristotelskou a scholastickou ontologii s myšlenkami Descarta a Leibnize, za použití eukleidovských axiomaticko-deduktivních principů.

Pro zkoumání otázek klasifikační povahy, tj. jaké třídy jsoucen existují a jaké jsou mezi nimi na nejobecnější úrovni typy vztahů, se v současnosti používá označení formální nebo deskriptivní ontologie či metafyzika. Ze současných myslitelů je pak v tomto ohledu významný zejména P. F. Strawson a jeho spis *Individuals* (1959). Strawson se domnívá, že lidské bytosti mají určité sdílené, univerzální konceptuální schéma, které nemůže být dále zdůvodněno nebo převedeno na základnější koncepty, může být pouze popsáno. Výsledkem takového popisu je přesná identifikace formálních ontologických struktur sdílených všemi lidmi. Každý jednotlivec disponuje časoprostorovým klasifikačně-identifikačním rámcem centrovaným na aktuálně vnímanou zkušenost. Strawson pak zkoumá proces identifikace nových jednotlivin a jejich začleňování a lokalizaci v rámci jedinečného individuálního klasifikačního rámce. Zaměřuje se zejména na ontologické kategorie hmotných těles a událostí, ale zkoumá i další možnosti, například rámec založený na čistě akustické zkušenosti. Strawsonův výzkum má zásadní vliv na vývoj základních ontologií v informatice a řada autorů z něj explicitně vychází. V oblasti filosofie pak pro ontologické inženýrství relevantním způsobem na Strawsona navazuje například D. Wiggins (2001).

V oblasti informatiky poprvé výraz ontologie použil pravděpodobně J. McCarthy v textu o nemonotónním odvozování (1980). Do souvislosti s informatikou dával ontologie v 80. letech i J. F. Sowa (1984) v kontextu tematiky modálních logik. Obvykle citovaná definice se v roce 1993 objevuje v textu T. Grubera (1993), který definuje ontologii jako „explicitní specifikaci konceptualizace". J. F. Sowa pak navrhuje v roce 2000 tuto definici: „Předmětem ontologie je studium kategorií věcí, které existují nebo mohou existovat v určité doméně. Výsledkem tohoto studia, nazývaným 'ontologie', je katalog věcí, jejichž existenci předpokládáme v dané doméně D, z perspektivy osoby používající jazyk L, aby mluvila o D" (Sowa, 2000). V tomto kontextu bývá také někdy doporučováno, aby pro označení vědecké disciplíny bylo používáno výrazu Ontologie, zatímco pro informační artefakt (např. soubor), který je výsledkem analýzy určité domény (popř. pro studium konkrétní domény), výrazu ontologie (Guarino, 1998), (Øhrstrøm, 2005).

1.2 Základní pojmy a definice

Výzkum ontologií se stal významným tématem iniciativy KSE (Knowledge Sharing Effort) agentury ARPA z počátku 90 let. Tato inciativa si stanovila za cíl rozvoj metod a infrastruktury pro sdílení a opakované využití existujících znalostí. Tím se zjednoduší a usnadní vytváření rozsáhlých znalostních systémů. V éře informačních technologií jsou znalosti nejdůležitější přidanou hodnotou, jejich vytváření je nákladné, a proto má jejich sdílení a opakované použití přímý ekonomický dopad.

Pokud vyjdeme z uvedené Gruberovy definice ontologie jako „explicitně popsané sdílené konceptualizace" (nějaké části reálného světa), vidíme, že charakterizuje některé důležité vlastnosti ontologie. Především, ontologie je konceptualizací. Vymezuje tedy, které objekty budeme brát v úvahu, co znamenají, jaké mají vlastnosti a v jakém jsou vztaku s jinými uvažovanými objekty. Tato konceptualizace je sdílená. To znamená, že existuje komunita, skupina řešitelů úloh, která se dohodla a přijala společnou konceptualizaci při řešení určité třídy úloh. Předpokládáme, že členy komunity jsou kognitivní agenti. Termínem „kognitivní agent" (krátce jenom „agent") budeme označovat jak počítačové programy, tak i lidi. Agenti existují v nějakém prostředí, získávají z něj informace a využívají je k ovlivňování tohoto prostředí, plánovitě v souladu se svými cíli. Komunita agentů muže být smíšená a obsahovat jak programy, tak i lidi.

Pokud mají agenti sdílet informace o okolním světě, musejí je nejen stejně interpretovat, ale musejí mít také prostředky pro vzájemnou komunikaci. Odtud vyplývá, že sdílená konceptualizace musí být explicitně popsaná, tj. musí existovat jazyk, kterým si lze znalosti předávat uvnitř komunity a kterému všichni její členové rozumějí stejně.

Aby ontologie zachycovala vlastnosti, které pokládáme intuitivně za významné pro další odvozování, musí charakterizovat třídy objektů intensionálně, tedy jejich vlastnostmi, a ne extensionálně, tj. vyjmenováním jednotlivých instancí. Tím je konceptualizace rozšířená na všechny instance, které splňují definiční vlastnosti. Guarino proto dodává, že ontologie je částečnou konceptualizací. Jednotlivé instance nejsou zahrnuty do ontologie, ale tvoří na ontologii závislou strukturu nazvanou báze znalostí. Podrobnější diskusi a formalizaci intensionálního popisu tříd lze najít v práci (Guarino, Giaretta, 1995).

Ontologie slouží jako nástroj pro reprezentaci a sdílení znalostí obsažených v konceptualizaci. Sdílení chápeme jako využití znalostí v rámci komunity, tedy v případě, kdy uživatel znalostí není jejich autorem. Časová závislost není podstatná, i když samozřejmě vytvoření znalostí musí předcházet jejich využití jiným agentem. Kromě sdílení však mají ontologie také velký význam v opakovaném použití znalostí (knowledge reuse) při řešení stejné nebo podobné úlohy.

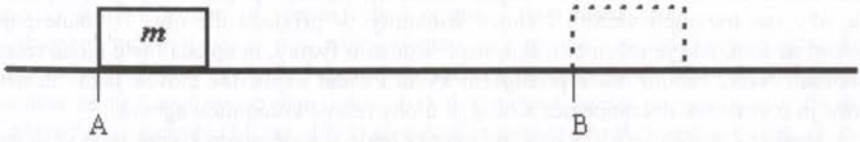
Všechny uvedené vlastnosti ovlivňují požadavky na návrh a implementace ontologií. Jednotlivé požadavky se navíc navzájem ovlivňují. Jiná konceptualizace je zapotřebí pro sdílení znalostí pouze mezi lidmi a jiná pro komunikaci smíšených kognitivních agentů. Zatímco člověk je někdy schopen nesprávnou nebo chybějící informaci opravit a mnohoznačnost odstranit na základě svých znalostí a implicitního kontextu, software tyto schopnosti většinou nemá. S tím také souvisí jazyk, který je potřeba použít pro reprezentaci znalostí. Pro sdílení informací v komunitě agentů je nutné použít formální jazyk s přesně definovanou sémantikou, zatímco pro řešení v komunitě lidí stačí neformálně

nebo semiformálně reprezentovaná konceptualizace, např. přirozeným jazykem nebo grafickými schématy. Pokud probíhá řešení ve smíšené komunitě softwarových agentů a lidí, je třeba také řešit, jak budou formálně definované pojmy lidé interpretovat. V obou případech však musí ontologie zahrnout všechny objekty a vlastnosti, které jsou zapotřebí pro řešení úlohy.

1.3 Konceptualizace

Konceptualizace je základním krokem při řešení úloh. Problémy, které řešíme, jsou totiž většinou „špatně strukturované" (ill-structured), což znamená, že není dostatečně popsán počáteční nebo cílový stav, nebo chybí kritérium, které má řešení splnit, popř. neexistuje výpočetně efektivní algoritmus řešení (Simon, 1973). Konceptualizace určí, které objekty budou v řešení uvažovány a jaké jsou jejich vlastnosti, a tím umožní vybrat vhodný formální aparát pro řešení úloh.

Uvažujme jednoduchý technický problém, znázorněný schematicky na obr. 1.1. Cílem je přemístit těleso tvaru kvádru posunutím po vodorovné podložce z místa A do místa B.



Obr. 1.1 Konceptualizace jednoduché úlohy.

V úloze se vyskytují tři objekty: (1) kvádr, který má být přemístěn, (2) podložka, na které těleso leží, a (3) prostředí, ve kterém se celá úloha řeší. Každé těleso v reálném světě má svoji délku, výšku a hloubku, rozložení hmoty v takto vymezeném trojrozměrném prostoru a fyzikální a chemické vlastnosti podle materiálu, z něhož je vytvořené. Pokud na ně působí vnější síly, bude v závislosti na vlastnostech materiálu vykazovat jak elastické, tak i plastické deformace. Podobně podložka a prostředí jsou také reálnými objekty, které mají své rozměry, fyzikální a chemické vlastnosti (tuhost, elasticitu, koeficient tření, odpor vzduchu, popř. kapaliny atd.). Konceptualizací určíme, které objekty a vlastnosti budeme při řešení uvažovat. Těleso může být například reprezentováno hmotným bodem nebo ideálně tuhým tělesem s danými rozměry, popřípadě objektem, který se elasticky nebo plasticky deformuje atd. Podložka může být tuhá nebo opět s možností deformace, tření mezi tělesem a podložkou můžeme vzít v úvahu nebo zanedbat. Prostředím může být vzduch, vzduchoprázdno, stlačený plyn, ideální nestlačitelná kapalina atd. Rozhodnout, které objekty a vlastnosti budeme uvažovat, je právě úlohou konceptualizace. Teprve po jejich vymezení lze nalézt vhodný formální aparát k řešení úlohy. Jednou z možností je například zanedbat fyzikální rozměry tělesa, tření a deformace a nahradit těleso hmotným bodem, který se pohybuje ve vzduchoprázdnu (zanedbáme tření a odpor vzduchu). Takto získaný popis umožní použít jistou třídu algoritmů, ale současně se tím definuje, jaké vlastnosti bude mít řešení. Nebude napří-

klad možné zkoumat množství tepla, která vznikne pohybem tělesa po podložce. U složitějších úloh se konceptualizace a následné řešení mohou opakovat – pokud nelze v rámci dané konceptualizace úlohu uspokojivě vyřešit, je třeba konceptualizaci změnit a zkusit nové řešení. Schön tento proces nazývá „problem reframing" (Schön, 2003).

Na uvedeném příkladu jsme ukázali, jak konceptualizace podmiňuje řešení úloh: Základem úlohy je určit vybrané části reálného světa, které na zvolené úrovni abstrakce chceme popsat, a definovat jejich interakce s okolím. Podobná metodika se používá i v jiných oblastech. Například v obecné teorii systémů se podobně definují systémy na objektech (Klir, 1985), stejný princip se uplatňuje v objektově orientovaném programování a designu (Booch, 1994), návrzích databází (Date, 1995) nebo řešení úloh v umělé inteligenci (Genesereth a Nilsson, 1978). Zjevně nejde o nic nového. Tento způsob řešení úloh nám připadá tak samozřejmý, protože ho používáme i při běžných „nevědeckých" činnostech každodenního života. To, že své problémy konceptualizujeme, často ani nevnímáme.

1.4 Jednoduchý model komunikace

Podmínkou pro sdílení znalostí v nějaké komunitě je, aby byl popis vyjádřen explicitně tak, aby mu rozuměli všichni členové komunity. V příkladu dle obr. 1.1 bude popis záviset na tom, zda je určen člověku, např. studentu fyziky, nebo zda bude úloha řešena robotem. Navíc robotu může přemístění kvádru zadat explicitně člověk jako cíl, nebo může jít o výsledek dekompozice složitější úlohy řešené komunitou agentů.

Interakce člověka se softwarovými agenty hraje v současném životě stále větší roli, a proto je sdílená konceptualizace řešeného problému důležitá. Agenty si nemusíme představovat jenom jako autonomní elektromechanické roboty. Pomocí internetu vyhledáváme dopravní spojení, vybíráme a kupujeme zboží, platíme účty. Řešení těchto úloh jsou často implementována agentním systémem. Jedním z nejznámějších příkladů je služba, kterou poskytuje internetové knihkupectví Amazon. Na základě předchozích objednávek si softwarový agent vytváří čtenářský profil svého zákazníka, snaží se pochopit jeho zájmy, porovnává je s předmětovým tříděním knih, které má v nabídce a podle výsledku mu nabízí nové tituly. Podobnou službu využívají běžně i velké obchodní domy a supermarkety. Podle momentální potřeby tak dynamicky vzniká heterogenní komunikační síť s mnoha účastníky, kteří vyžadují a poskytují si navzájem nejrůznější služby. Probíhá nejen komunikace mezi lidmi a počítači, ale i vzájemná komunikace mezi softwarovými agenty. Vzhledem k vhodně navrženému uživatelskému prostředí si člověk často neuvědomuje, jak je jeho požadavek dále rozpracován a řešen.

Klíčovým problémem komunikace je porozumění a pochopení smyslu sdělení. Základní principy lze ukázat na jednoduchém modelu jazyka. Jazyk se skládá z množiny výrazů L, z množiny významů (pojmů, konceptů) M a ze zobrazení σ: $L \rightarrow M$, nazvaného sémantika, které přiřazuje výrazům z L významy z M. Při komunikaci předpokládáme základní jazykovou konvenci: Sdělujeme si výrazy, ale máme na mysli významy, ke kterým odkazují. Tuto konvenci můžeme v případě potřeby porušit, ale musíme to explicitně vyznačit. Například řekneme-li, že „stůl" má 4 písmena, máme na mysli výraz (slovo „stůl") nikoliv význam (nábytek, který má alespoň 3 nohy a desku, na kterou lze

něco položit). Odkaz na výraz musíme vyznačit, v tomto případě dáváme slovo „stůl" do uvozovek. Sémantika nemusí být (jednoznačnou) funkcí, jedinému výrazu může být přiřazeno několik významů. Lze tedy psát σ(výraz) = význam-1, σ(výraz) = význam-2, význam-1 $\neq$ význam-2. Tento jev se nazývá polysémie nebo homonymie. Oba pojmy se často zaměňují, přestože je mezi nimi významný rozdíl. Polysémie označuje vztah, kdy dvě slova mají rozdílný význam, ale jsou etymologicky příbuzná. Například výraz „jazyk" může znamenat svalový orgán umístěný v dutině ústní, část boty zakrývající nárt nebo prostředek sloužící ke komunikaci. Ve vyjmenovaných případech jsou významy zcela rozdílné, ale existují jasné etymologické závislosti: jazyk v botě získal zřejmě svůj název, protože je tvarem podobný jazyku v ústní dutině, a jazyk jako mluvená forma komunikace souvisí s pohybem stejnojmenného svalového orgánu v ústech. Naproti tomu v případě homonymie etymologická příbuznost chybí. Příkladem může být slovo kolej, v odlišných významech jednak jako ubytovna vysokoškolských studentů, jednak jako rýha vyjetá vozem v blátivé cestě. V prvním případě je význam odvozený z latinského collegium, znamenající komunitu nebo spolek, zatímco druhý význam má původ ve slovanském slově kolo – jde o stopu způsobenou kolem vozu. Tyto významy spolu vůbec nesouvisí. Můžeme nalézt ještě třetí význam slova „kolej" (slovník připouští „kolej" i „koleje") jako dvojici ocelových profilů – kolejnic, které slouží k jízdě vlaku. Snadno rozpoznáme, že jde o homonymii s prvním případem (studentskou ubytovnou), ale o polysémii s druhým případem (rýhou vyjetou kolem vozu).

Podobně může platit, že pro jediný význam existuje více výrazů, tedy σ(výraz-1) = σ(výraz-2), výraz-1 $\neq$ výraz-2. Tento jev se nazývá synonymie a výrazy se stejným významem se nazývají synonyma. Jako příklad synonym mohou sloužit slova „hezký" a „pěkný". V literatuře (Miller, 1998) se množina synonym někdy označuje jako synset a vyznačuje se vyjmenováním významově ekvivalentních výrazů ve složených závorkách. Uvedený synset bychom zapsali jako {hezký, pěkný}. Homonymie, polysémie a synonymie jsou znakem bohatosti a krásy jazyka. Při snaze o přesnou interpretaci sdělení však mohou působit potíže, zejména když příjemcem sdělení není člověk. K odstranění významové mnohoznačnosti lze jak v mezilidské komunikaci, tak i při komunikaci se agentem použít kontext, ve kterém je výraz použit, nebo lze nejprve předpokládat obvyklou (implicitní, default) hodnotu a problém mnohoznačnosti řešit až teprve, když se projeví jeho důsledky.

Na množině významů lze definovat relaci subsumpce (označovanou obvykle jako relace is-a) a využít ji v konstrukcích typu význam-1 $\subset$ význam-2, popř. σ(výraz-1) $\subset$ σ(výraz-2). Symbol $\subset$ zde označuje subsumpci – pojmovou podřazenost. Pojem A subsumuje pojem B, značíme $B \subset A$, pokud z jejich definic plyne, že každá entita patřící pod B musí patřit také pod A. Potom význam-1 nazýváme hyponymem významu-2 a význam-2 nazýváme hyperonymem významu-1. Vzhledem k základní jazykové konvenci můžeme také říkat, že výraz-1 je hyponymem výrazu-2 a výraz-2 je hyperonymem výrazu-1. Sémantický lexikon WordNet (Miller, 1998) označuje relace hyperonymie symbolem @→ a hyponymie symbolem ~→, takže lze pro uvedenou relaci psát výraz-1 @→ výraz-2 a výraz-2 ~→ výraz-1. V literatuře o ontologiích se však nejčastěji používá grafické označení obvyklé ze softwarového inženýrství, tedy např. výraz-1 → výraz-2, kde šipka směřuje k hyperonymu (nadtřídě). Subsumpce je tranzitivní asymetrická relace, a proto můžeme pomocí ní pojmy řetězit a vytvářet hierarchické taxonomie. V litera-

tuře o ontologiích bývá zvykem v souladu s počítačovou terminologií označovat hyperonyma a hyponyma jako třídy a podtřídy, popř. nadtřídy. Subsumpce je relací velmi významnou a často používanou při popisu konceptualizace, a proto se k ní při úvahách o různých formách konceptualizace ještě vrátíme.

Protože je subsumpce sémantickou relací a vstupují do ní tedy významy, mohou být pojmy v hierarchii reprezentovány pomocí celých synsetů. Například hierarchii pojmů z živé přírody popisuje posloupnost {rehek, čermák} @→ ptáci @→ {živočichové, živí_tvorové}. Tento příklad může současně sloužit jako ukázka různých konceptualizací stejné domény. Biologie totiž pojem čermák nezná, jde o staré lidové označení rehků. Uvedená hierarchie tedy popisuje českou přírodu způsobem, který používá a sdílí tradiční lidová kultura. Biologie jako vědní disciplína má jinou, systematickou konceptualizaci s dělením na pojmy: Říše/Čeleď/Kmen/Podkmen/Třída/Podtřída/Řád, ve které řadí pojmy od obecných ke speciálním. V uvedeném případě uvádí biologie taxonomii živočichové ⟶ strunatci ⟶ ... ⟶ drozdovití ⟶ rehek domácí. Tato konceptualizace je sdílená odbornou komunitou biologů. Obě konceptualizace jsou platné, přestože se liší. Jsou totiž určené pro odlišné komunity.

Potřeby a možnosti uživatelů ontologie také určují, do jaké míry má být ontologie formalizována. Pokud bude sdílena v rámci skupiny lidí, lze ontologii popsat přirozeným jazykem. Formální vyjádření pomocí axiomů a pravidel, které lze na pojmy aplikovat, většinou není zapotřebí. Lze je nahradit a ontologii doplnit všeobecnými znalostmi (commonsense knowledge), které lze u uživatelů předpokládat. Pokud má ale ontologie podporovat počítačové zpracování znalostí, měla by být reprezentována tak, aby program nemusel potřebné informace interpretovat složitým zpracováním přirozeného jazyka. V tom případě je ontologie reprezentována vhodným formálním jazykem, který většinou vychází z jazyka predikátové logiky (Antoniou, van Harmelen, 2008), (Gruber, 1993), (Genesereth, Fikes, 1992), (Sowa, 2000).

Nejjednodušším příkladem konceptualizace, která je popsaná přirozeným jazykem, jsou glosáře v odborných knihách. Semiformální konceptualizace se používá v tezaurech a řízených slovnících k indexování databází dokumentů. Vytvoření glosářů a tezaurů je také často výchozím krokem při návrhu ontologií. Proto je těmto dvěma metodám konceptualizace věnována následující část výkladu.

1.5 Glosář a tezaurus

Jednoznačná interpretace psaného textu, zejména odborných termínů a pojmů, je důležitá pro pochopení obsahu každé publikace. V odborných textech, zejména knihách a výukových materiálech, bývají pojmy nové nebo pojmy, jejichž jednoznačné chápání je důležité pro porozumění textu, vysvětleny v glosáři (glossary). Například monografie (Strachan a Read, 1999) z oblasti genetiky uvádí tuto definici alely:

Alela: *Jedna z několika možných forem genu nebo sekvence DNA vyskytující se na specifickém chromozomálním lokusu.*

Podobně je zde vysvětleno mnoho dalších pojmů z genetiky. Český glosář genetických termínů na úrovni vhodné pro studenty středních a vysokých škol lze nalézt na

http://genetika.wz.cz. V glosářích jsou pojmy většinou vysvětleny textem v přirozeném jazyce, neboť se očekává, že ho bude číst a interpretovat člověk. Předpokládá se také určitá úroveň znalostí pojmů. Například v monografii (Strachan a Read, 1999) nejsou v glosáři vysvětleny pojmy DNA a gen, na rozdíl od uvedeného českého glosáře, který je určen pro širší okruh čtenářů se všeobecným vzděláním. Cílem glosáře je informovat čtenáře, přičemž se předpokládá, že je schopen porozumět napsanému vysvětlení. Pojmy jsou většinou seřazeny abecedně a není nad nimi explicitně definována žádná další formální struktura. Přesto by definice v glosáři měly splňovat některé podmínky. Pojmy by měly být definovány intensionálně, tj. svými vlastnostmi, a ne extensionálně, tj. výčtem instancí (viz též oddíl 1.2). Intensionální definice bývá vyjádřena pomocí hyperonyma a podmínkami, které vybírají některé z možných hyponym, a tak blíže určují pojem. Tento způsob definování pojmů sahá až k Aristotelovi. Říkáme, že definice využívá genus (nadtřídu, širší kategorii, hyperonymum) a differentiae (kritérium, znaky, které blíže rozlišují hyponymum, a tak definují podtřídu). Příkladem může sloužit definice uvedená na http://genetika.wz.cz.

Homozygot: *Organismus, jehož obě alely na sledovaném lokusu (pro příslušný gen) jsou stejné.*

Tato definice pojmu homozygot má dvě části: genus („organismus") a differentia, tj. podmínku, kterou musí organismus splňovat, aby byl homozygotem („alely sledovaného genu musí být stejné"). Podobně definujeme pojem heterozygot, genus je stejný, jen differentiae se liší (alely sledovaného genu musí být různé).

Heterozygot: *Organismus, jehož obě alely na sledovaném lokusu (pro příslušný gen) jsou různé.*

Další podmínkou, kterou by definice v glosáři měly splňovat, je neexistence cyklů. Definice by měly mít stromovou strukturu s tím, že kořeny stromu jsou vysvětleny pomocí pojmů, o kterých předpokládáme, že jsou známé.

Strukturovanějším popisem důležitých pojmů z dané problémové oblasti je tezaurus, který se používá například k anotování a indexování dokumentů. Tezaurus je vlastně řízený slovník pojmů z dané oblasti a navíc definuje nad množinou pojmů některé základní relace. Cílem anotace je umožnit budoucímu čtenáři najít snadno relevantní informaci. V knihovnictví bývá seznam indexů často označován jako předmětový rejstřík. Komunitu uživatelů tvoří jednak lidé (čtenáři, anotátoři, knihovníci, pracovníci v informačních technologiích), jednak programy, včetně softwarových agentů, které celý proces zpracování informace podporují.

Pro strukturu tezaurů existují doporučení a standardy, např. ISO 2788-1986. Kromě struktury definuje norma také jednoduchý jazyk, který slouží k popisu pojmů a struktury. Vzhledem k tomu, že základním použitím tezaurů je anotace odborného textu, je snahou dosáhnout jednoznačnosti anotace a současně zabezpečit pokrytí co největší části předpokládané odborné oblasti. K tomu lze využít synonym. Protože jde práci s výrazy, nezáleží na tom, zda bude text anotovat profesionální anotátor nebo program. Pojem „synonymum" je ale v normě ISO 2788-1986 chápán poněkud volněji. Aby bylo možné stejným tezaurem anotovat velké množství dokumentů, obětuje norma přesnější definice pojmů. Z vybrané množiny slov, která označují přibližně stejný pojem, je vždy vybráno

jedno jako doporučený termín, který reprezentuje celou skupinu. V normě je označen jako *preferred term*, v některých tezaurech je označován jako *GO term*, v češtině se používá i pojem *deskriptor*. Ostatní slova dané skupiny chápaná jako synonyma, jsou označena jako nedoporučené termíny (non-preferred term nebo STOP term, česky též *nedeskriptor*). GO a STOP bývá zkráceno jako (G) a (S). Pojem doporučený a nedoporučený termín se týká anotátora, nikoliv autorů textu. Například v tezauru *Social Care Online: Subject Taxonomy* (2006) existuje skupina terminologických výrazů: emotional abuse (G), emotional neglect (S), mental cruelty (S), z nichž první byl vybrán jako doporučený a zbývající dva jako nedoporučené. To znamená, že když se kdekoliv v textu určeném k anotaci vyskytne libovolný z těchto tří termínů, anotátor označí tuto část textu doporučeným indexem „emotional abuse". Bohatost textu není ovlivněna, autor textu není nijak omezen, a přitom anotace dovoluje s dostatečnou přesností najít hledaný dokument. Z uvedeného příkladu je zřejmé, že skupina pojmů, chápaných jako ekvivalent a anotovaných stejným termínem, netvoří vždy synonyma v přísném lingvistickém smyslu. Může se jednat například o kvazisynonyma (ISO 2788-1986), tedy výrazy označující sice různé pojmy, které však pro indexování lze považovat za rovnocenné. Příkladem může být dvojice pojmů „elektrický odpor" a „elektrická vodivost". Tyto veličiny jsou jistě rozdílné, jedna je převrácenou hodnotou druhé, ale popisují stejnou vlastnost materiálu – schopnost vést elektrický proud. Jiným příkladem povolené sémantické nepřesnosti je použití jména třídy na indexování podtříd, ale i instancí. Například vyvřelé, usazené, přeměněné horniny, žulu, pískovec a břidlici označíme společným pojmem horniny. Norma zavádí pro přiřazení doporučených a nedoporučených termínů zkratky USE a UF (use for). Vysvětlující poznámky v přirozeném jazyce se označují SN (scope notes).

Uvedený příklad hornin by byl tedy popsán takto:

Horniny (G)
UF vyvřelé horniny
UF usazené horniny
UF přeměněné horniny
UF žula
UF břidlice

a dále

Vyvřelé horniny (S)
USE horniny
Usazené horniny (S)
USE horniny
Přeměněné horniny (S)
USE horniny
Žula (S)
USE horniny
Břidlice (S)
USE horniny.

Užití jména třídy jako doporučeného termínu (G) při indexování jako jména podtřídy je zjednodušením za cenu sémantické nekorektnosti, která je však při práci s tezaury přípustná. Cílem tezauru totiž není přesně vymezit význam pojmu, ale vytvořit jednoduchý prostředek k nalezení požadovaného dokumentu. V ontologiích by byl takový po-

stup považován za chybný. I v tezaurech však bývá častější, že se k vytvoření hierarchie pojmů používá relace třída–podtřída. V textové reprezentaci zavádí norma označení BT (broader term, hyperonymum) pro odkaz na hierarchicky vyšší pojem a NT (narrower term, hyponymum) pro odkaz na hierarchicky nižší pojem. Uvedený příklad lze tedy lépe reprezentovat hierarchií

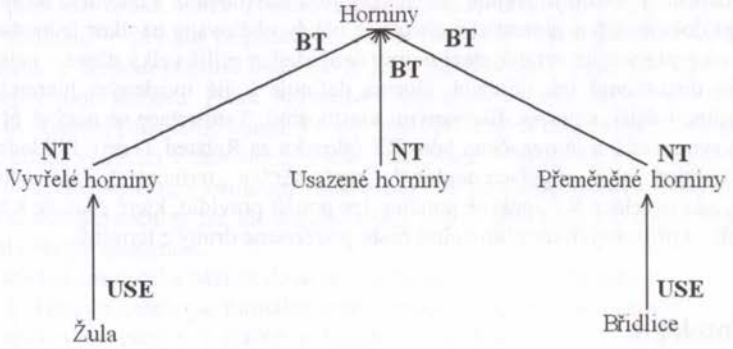
Horniny
NT vyvřelé horniny
NT usazené horniny
NT přeměněné horniny

a dále

Vyvřelé horniny
BT horniny
Usazené horniny
BT horniny
Přeměněné horniny
BT horniny.

Žula by mohla být reprezentována buď jako hierarchicky nižší termín (NT) než *Vyvřelé horniny*, nebo jako nedoporučený termín (S) s tím, že by odpovídajícím (G) termínem byly *právě vyvřelé horniny*. Podobný vztah by platil mezi pojmy *Břidlice* a *Přeměněné horniny*.

Grafické zobrazení možné reprezentace v tezauru je uvedeno na obr. 1.2.



Obr. 1.2 Hierarchie hornin.

V tezauru se kromě subsumpce, označované také jako relace is-a, používá i jiných relací k vytváření hierarchií. Může to být mereologická relace, tedy vztah celku a částí nebo vztah třídy a instance, který jsme již uvedli v souvislosti s doporučenými a nedoporučenými termíny. Příkladem hierarchie založené na vztahu celku a částí jsou systémy a orgány živých organismů (kardiovaskulární systém, žíly, srdce, tepny), geografické pojmy (kontinent, stát, město, ulice, náměstí) nebo organizační struktury (univerzita, fakulty, katedry, laboratoře).

Norma (ISO 2788-1986) uvádí jednoduchý způsob, označovaný jako test „všechny–některé", kterým lze ověřit, zda v hierarchii opravdu jde o subsumpci. Pokud lze říci, že všechny objekty patřící do podtřídy jsou také objekty nadřazené třídy a některé objekty nadřazené třídy jsou objekty podtřídy, jde o vztah třída–podtřída. Subsumpci jsme

v oddílu 1.3 definovali jako množinovou inkluzi významů, v oddílu 1.10 ukážeme, jak se zapisuje jednoduchou formulí jazyka predikátové logiky. V příkladu dle obr. 1.2 platí, že všechny vyvřelé horniny jsou horniny a některé horniny jsou vyvřelé horniny – vyvřelé horniny jsou tedy podtřídou hornin. Naopak, nelze říci, že všechny státy jsou kontinenty, všechna města jsou státy nebo že všechny fakulty jsou univerzitami. Proto v tomto případě nedje o subsumpci. V tezauru je ale definice hierarchie BT/NT pomocí mereologické relace přijatelná, lze tedy psát

Česká republika
    NT Praha
Praha
    BT Česká republika
Praha
    NT Karlovo náměstí
Karlovo náměstí
    BT Praha,

přestože nejde o hierarchii is-a. Použitím této hierarchie ovšem ztrácíme silnou vlastnost subsumpce, kterou je možnost dědění atributů. Z toho, že má Česká republika hranici s Rakouskem tedy nevyplývá, že má Praha hranici s Rakouskem. Subsumpce se ale v tezaurech většinou nevyužívá.

Z uvedeného výkladu je zřejmé, že tezaury jsou navrhovány s ohledem na snadnost indexování dokumentů a sémantická čistota je někdy obětována na úkor jednoduchosti. Proto ani na reprezentaci vztahů mezi pojmy není kladen příliš velký důraz – nejsou pro indexování dokumentů tak důležité. Norma definuje k již uvedeným hierarchickým relacím jedinou další, s nespecifikovanými vlastnostmi. Tato relace se nazývá příbuzný termín, je symetrická a je označena jako RT (zkratka za Related Term). Příkladem příbuzných termínů jsou „regulace teploty", „teploměr" a „termostat". Jako heuristiku k ověření, zda je relace RT správně použita, lze použít pravidlo, které říká, že k vysvětlení jednoho z příbuzných termínů velmi často používáme druhý z termínů.

1.6 Ontologie

Ontologie jsou dalším krokem směrem k přesnějšímu vymezení sémantiky pojmů. Glosáře definují neformálně přirozeným jazykem pojmy z dané oblasti, protože je uživatelem člověk vybavený dostatečnými znalostmi na to, aby definice správně interpretoval a použil. Tezaury vytvářejí z pojmů jednoduché struktury, které bude používat jak člověk, tak i strojový algoritmus. Typické úlohy podporované pomocí tezaurů jsou poměrně jednoduché. Spočívají ve vyhledávání informačních zdrojů podle zadané specifikace. Struktury v tezaurech umožňují částečně postihnout sémantiku dotazu a neomezovat se při hledání na pouhé srovnávání výrazů. Přesnost vyjádření významů je obětována kvůli jednoduchosti, a proto pro složitější úlohy tezaury nestačí.

Ontologie jdou ve vymezení sémantiky podstatně dále. Cílem není pouze zprostředkovat komunikaci mezi agenty a odstranit veškeré mnohoznačnosti, ale navíc zachytit znalosti potřebné pro řešení úloh z dané oblasti. Přitom není tak důležité zaznamenat

vyřešení jednoho jedinečného případu, ale poskytnout aparát pro řešení celé třídy navzájem podobných úloh. Proto se odlišují obecné znalosti, které jsou reprezentovány v ontologiích pomocí tříd a jejich vztahů, od jedinečných znalostí vyjádřených na úrovni instancí těchto tříd. Ty potom bývají zahrnuty do báze znalostí. Mezi ontologií a příslušnou bází znalostí existuje jednoznačný vztah. Z toho současně vyplývá, že k jediné ontologii může existovat několik různých bází znalostí, které popisují rozdílné případy. Dříve uvedené tezaury a glosáře rozdělení na třídy a instance nepoužívají.

Ontologie je formálně vyjádřena jako hierarchická struktura tříd, které popisují uvažované pojmy. Hierarchie je tvořena relací is-a (subsumpcí). Jak ale ukážeme, na vlastnosti tříd, které vstupují do této relace, klademe přísnější požadavky. Třídy mohou být dále popsány pomocí svých atributů (properties, slotů) a axiomů, které musejí splňovat. Instance potom přiřazují atributům konkrétní hodnoty. Třídy mohou definovat pro hodnoty atributů různá omezení (facety, restrictions), např. typ hodnoty, minimální a maximální kardinalitu atd.

Formální specifikace má za důsledek, že jsou explicitně vyjádřeny všechny předpoklady, přijaté při konceptualizaci, ty tedy již nejsou součástí kódu zpracujícího programu. Tím se stávají snadno dostupné k dalšímu zpracování jinými programy. Všechny předpoklady, které určují zvolenou konceptualizaci, vymezují ontologický závazek (ontological commitment), který chápeme jako vztah mezi reálným světem a jeho konceptualizací (podrobnější výklad najdeme v dalším textu). Budeme-li například zkoumat pád fyzikálního tělesa, může zvolená konceptualizace buď ignorovat, nebo uvažovat odpor vzduchu. V prvním případě bude na těleso působit jen gravitační síla směrem shora dolů, v druhém případě bude navíc ještě působit síla vyjadřující odpor vzduchu a směřující zdola nahoru. Práce vykonaná touto silou se přemění v teplo a lze proto uvažovat oteplení padajícího tělesa. To v prvním případě zvolená konceptualizace nedovoluje, odpor vzduchu v takovémto světě „neexistuje". Stejný název – ontologický závazek – se také používá k označení vztahu mezi řešícím algoritmem a použitou ontologií. Tuto mnohoznačnost lze ale snadno rozpoznat z kontextu, a proto jí není potřeba věnovat zvláštní pozornost.

Používání ontologií a bází znalostí při řešení úloh má tyto důsledky:

1. Tím, že ontologie formálně reprezentuje znalosti z dané oblasti, se umožňuje jejich opakované použití v jiných aplikacích. Například jednou vytvořenou ontologii času popisující časové intervaly, body na ose času a jejich absolutní i relativní vztahy (Allen, 1984), (Matoušek, Kouba, 2006) lze znovu použít v dalších aplikacích, které sdílejí použité chápání časových pojmů. V současnosti již existuje velmi často reálná možnost, že ontologii, kterou potřebujeme, již někdo vytvořil a není třeba ji složitě vyvíjet. To je také jedním z prvních doporučení metodiky pro návrh ontologií, kterou uvedeme později. Tak se naplňuje myšlenka sdílení znalostí, ale současně vznikají úlohy týkající se mapování a spojování ontologií. Tato filosofie je nejdále dovedena v aplikacích sémantického webu, jak ukážeme později.

2. Lze snadněji oddělit znalosti popisující problémovou oblast od znalostí, které se týkají metod jejich zpracování. Protože však specifikace a metody řešení úloh jsou také oblastmi, které lze konceptualizovat (obsahují například pojmy jako kritérium, počáteční stav, cílový stav, možné akce, omezení, požadavky atd.), lze je také popsat ontologií, která potom bude explicitně specifikovat jednak dostupné znalosti, jednak

znalosti potřebné k nasazení algoritmů řešení. Světem, který popisujeme, je potom svět možných úloh a algoritmů řešení a ontologický závazek říká, které objekty v tomto světě smíme použít k řešení. Spojením instance vhodného řešícího algoritmu, specifikace úlohy a doménové ontologie, resp. příslušné báze znalostí vzniká úplná aplikace. Explicitní vyjádření znalostí v ontologiích umožní ověřit, zda vybraný algoritmus opravdu řeší úlohu s danou specifikací a zda znalosti, které algoritmus potřebuje, jsou k dispozici v předpokládané doménové ontologii. O otázkách architektury založených na spojení ontologie úlohy, ontologie řešících algoritmů a doménových ontologiích diskutuje podrobně (Motta, 1999). Lze oprávněně namítnout, že takto vzniklá aplikace bude výpočetně nejspíš velmi neefektivní. Poslouží ale dobře jako počáteční model řešení, který ověří, zda jsou splněny podmínky vedoucí k řešení úlohy.

3. Příbuzné ale vzájemně odlišné konceptualizace lze porovnávat tím, že vytvoříme zobrazení mapující ontologie navzájem. Například dva zásilkové obchody s podobným sortimentem zboží mají katalogy s rozdílnou klasifikací zboží. Svůj sortiment mohou porovnat mapováním ontologií, kterými se katalogy řídí. Nalezení korespondence mezi odpovídajícími pojmy není snadnou úlohou, protože se musí opírat o vyjádření sémantiky. V jednom katalogu mohou být pily na dřevo vedené jako podtřída zahradnického nářadí, ve druhém jako nástroje pro kutily. Protože ontologie jsou sémantické reprezentace, je mapování ontologií úlohou, kterou musíme řešit na sémantické úrovni. Stručně se k tomuto problému vrátíme později.

4. Použití ontologií má i některé technické důsledky. V mnoha případech totiž omezuje nároky na přenášená data a tím i na potřebnou šířku pásma, které je nutné pro přenos dat. Objekty tříd mohou být popsány atributy s přiřazenými výchozími hodnotami a přenášet se musí pouze hodnoty, které se od výchozích liší. Agent na přijímací i vysílací straně používá stejnou ontologii a tu lze využít k interpretaci přenesených dat.

1.6.1 Jednoduchý příklad

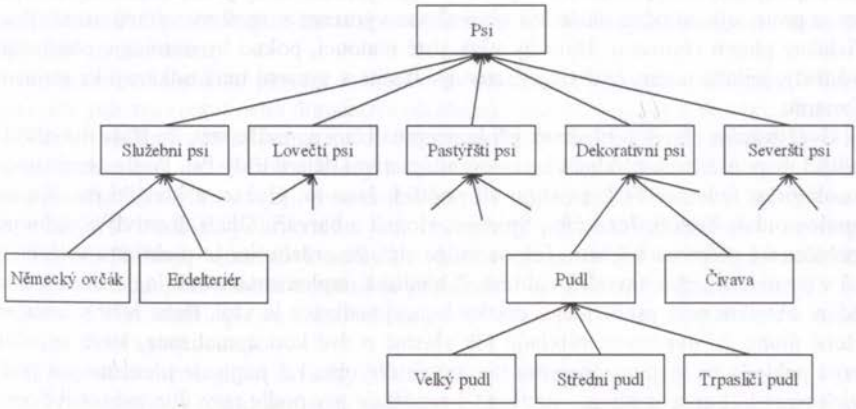
Jako příklad uvedeme kynologickou ontologii popisující plemena psů. Tato ontologie bude mít jako nejvyšší třídu psi s podtřídami služební psi, dekorativní psi, lovečtí psi, pastýřští psi a severští psi. V tomto dělení má genus označení psi a differentia pak využití. Třída služební psi se dále dělí na podtřídy německý ovčák, erdelteriér a další. Třída dekorativní psi má podtřídy pudl, čivava atd. Na této úrovni určuje differentia vzhled. Třída pudl má podtřídy velký pudl, střední pudl, toy pudl a trpasličí pudl. Differentia je velikost psa. Třída psi, může mít atributy: jméno, barva srsti, výška, pohlaví, chovatel a majitel. Některé z těchto atributů mají u jednotlivých tříd definovaná omezení. Například výška u třídy střední pudl je omezena hodnotami min-value = 35 cm a max-value = 45 cm. Grafické znázornění této ontologie vidíme na obr. 1.3.

Třídy ontologie popisují obecné vlastnosti platné pro všechny jejich členy. Třída střední pudl tedy charakterizuje všechny střední pudly. O tom, že je hierarchie opravdu vytvořená relací is-a, se lze přesvědčit testem „všechny–některé": Každý služební pes je pes a některý pes je služební pes. Podobně každý německý ovčák je služební pes a některý služební pes je německý ovčák atd.

Na uvedeném příkladu můžeme demonstrovat několik otázek, které jsme museli při návrhu této ontologie rozhodnout.[4]

[4] K řadě těchto otázek se vrátíme v kapitole 4 věnované ontologickému inženýrství v prostředí sémantického webu, kde bude jejich řešení, s ohledem na použití konkrétního jazyka, často jednoznačnější.

Jednou z prvních otázek je, které objekty mají být reprezentovány jako třídy a být součástí ontologie a které mají být jako instance umístěné v bázi znalostí. Guarino (1997) ukazuje, že ontologie a báze znalostí (soubor tvrzení o instancích) by měly zůstat oddělené. Zatímco ontologie popisuje obecné charakteristiky tříd objektů, báze znalostí vyjadřuje vlastnosti jedinečných objektů, které definujeme v rámci zvolené konceptualizace. V literatuře lze však často najít případy, kdy je jako ontologie označována hierarchie tříd spolu s jejich instancemi. S rozdělením na třídy a instance souvisí rozhodnutí,



Obr. 1.3. Hierarchie tříd.

do jaké hloubky má pokračovat hierarchie tříd a od které úrovně mají být objekty reprezentovány instancemi. Zde existují dvě různá hlediska:

1. Pokud tvoří podřazené pojmy (hyponyma) přirozené třídy, měly by být také jako třídy reprezentovány. (Střední pudl by měl být reprezentován jako podtřída třídy Pudl.)

2. Záleží ale také na úloze, pro kterou je ontologie vytvářena. Nejspecifičtější pojmy by měly být reprezentovány jako instance. Budeme-li tedy hovořit o konkrétním středním pudlu, bude reprezentován jako instance třídy Střední pudl. Pokud budeme ale úlohou srovnávat vlastnosti jednotlivých plemen, a konkrétní psi nás nezajímají, může být Střední pudl instancí třídy Pudl. Ztrácíme tím ale možnost ontologii v případě potřeby dále prohlubovat.

Podtřídy by měly reprezentovat pojmy na stejné úrovni abstrakce. To lze posoudit podle kritéria, které specifikuje podtřídy (differentia). Toto kritérium by mělo být stejné pro všechny přímé podtřídy, které by se lišily pouze jeho hodnotou (Noy, McGuinness, 2001). Znakem nevhodného návrhu hierarchie by například bylo, kdyby třída pudl měla podtřídy velký pudl, střední pudl a černý pudl. Uvedené podtřídy pudla se liší velikostí psa. Podtřídy tedy mají některá omezení, která jejich nadtřída nemá. Může to být přidaná vlastnost nebo omezení hodnoty existující vlastnosti (např. Střední pudl má vymezenou nejmenší a největší velikost).

Pokud existuje v ontologii k dané třídě jediná podtřída, differentia buď nic nerozlišuje, protože kritérium „všechny–některé" je splněno triviálním způsobem („všechny–

všechny") a není zřejmé, proč byla podtřída definována, nebo vymezuje pouze jednu podtřídu, ale neříká nic o zbytkové podtřídě, tj. o instancích, které podmínku podtřídy nesplňují. Takovýto případ se pokládá za důsledek buď nevhodné konceptualizace, nebo neúplnosti ontologie (Noy, McGuinness, 2001).

Je třeba vybrat vhodná jména pro třídy. Pokud je ontologie plně formalizovaná, je význam třídy vyjádřen soustavou logických formulí, které musí každá instance třídy splňovat, a jediné, co žádáme, je, aby jméno třídy bylo v rámci ontologie jedinečné. Přesto je výhodné, když jméno třídy naznačuje význam. Třídy jsou sémantickým objektem, a proto jejich jméno může být libovolným výrazem z množiny výrazů označujících příslušný pojem (synsetu). Bylo by však jistě matoucí, pokud by ontologie obsahovala dvě třídy, jejichž jména by byla synonymy. Jména v synsetu totiž odkazují ke stejnému významu.

Uvažováním různých hledisek při konceptualizaci se může stát, že třída má několik nadtříd. K popsanému příkladu lze uvést alternativní dělení třídy Psi. Podle Mezinárodní kynologické federace FCI, existuje 10 podtříd. Jsou to: Ovčáci a honáčtí psi, Knírači a malossoidní, Teriéři, Jezevčíci, Špicové, Honiči a barváři, Ohaři, Retrívři a vodní psi, Společenská plemena a Chrti. Tak se může stát, že erdelteriér je podtřídou služebních psů v první hierarchii a teriérů v druhé. Z hlediska implementace ontologie sice násobné dědění obvykle není problémem, otázky konceptualizace je však třeba řešit v kontextu řešené úlohy. V uvedeném příkladu jde vlastně o dvě konceptualizace, které vyjadřují různé pohledy na stejnou doménu: hierarchie dle obr. 1.3 popisuje plemena psů podle jejich použití, konceptualizace podle FCI rozděluje psy podle rasy. Jde tedy o dvě ontologie, které mohou existivat nezávisle vedle sebe.

Uvedený příklad je ukázkou problému, který může vzniknout při pojmenování tříd. V ontologickém inženýrství bývá zvykem používat jméno třídy v singuláru, v některých doménách ale existuje již jiná ustálená konvence. V kynologii nalezneme jméno třídy „Služební psi" a ne „služební pes". V systematické zoologické taxonomii Nadříše, Říše, ..., Kmen, ...,Třída, ..., Řád, ..., Čeleď, Rod, Druh, ... se používá plurál jmen tříd od nejvyšší taxonomické kategorie Nadříše až po kategorii Čeleď a v singuláru třídy kategorie Rod a všechny podtřídy. Je tedy Řád Primáti (Primates), Čeleď Lidé (Hominidea), ale Rod Člověk (Homo) a Druh Člověk rozumný (Homo sapiens). Bylo by jistě kontraproduktivní měnit v ontologiích zažité zvyklosti, a proto je třeba k otázkce pojmenování tříd přistupovat individuálně. O taxonomii, která se používá v zoologii, se ještě zmíníme v odstavci 1.10.3 pojednávajícím o metodice DILIGENT.

V tomto jednoduchém případě jsme mohli sestavit ontologii pouze na základě intuice. Pro systematický návrh však existuje řada propracovaných metodik, které budou uvedeny dále.

1.7 Klasifikace ontologií podle principů modelování

Ontologie mohou být klasifikovány na základě některých významných vlastností (Cimiano, 2004). První takovou vlastností je operacionalizace, která rozlišuje ontologie hrubé (coarse-grained) a jemné (fine-grained). Hrubé ontologie se vyznačují použitím jazyka s malou expresivitou a malým počtem axiomů. Jemné ontologie naopak využívají

vysoce expresivní modelovací jazyk a zahrnují obvykle velké množství axiomů. Jemné ontologie přitom mohou přinášet větší nároky na odvozování. Někdy se také jemné ontologie, s ohledem na jejich typické použití, označují jako referenční či vývojové (development-time), zatímco hrubé jako sdílené (shared) či provozní (run-time) (Guarino, 1998b).

Další důležitou vlastností je expresivita[5], na jejímž základě dělíme ontologie na těžké (heavy-weight) a lehké (light-weight). Těžké ontologie vyjadřují ontologické závazky explicitně a obsahují rozsáhlou axiomatizaci. Lehké ontologie vyjádření ontologických závazků neobsahují a často se soustředí jen na ty oblasti, které jsou autory považovány za relevantní.

Významnou vlastností je také specifičnost ontologie. Ontologie může být obecná (generic), jádrová (core) nebo doménová (domain). Obecné ontologie se věnují konceptům, které přesahují rozsah jednotlivých domén (např. proces, událost), jádrové modelují centrální koncepty určité širší domény a doménové ontologie obsahují koncepty specifické pro danou, většinou úžeji vymezenou doménu. Základní ontologie, o kterých bude podrobněji řeč v následujícím oddílu, jsou obecnými, ovšem ne každá obecná ontologie je základní. Existují i doménově nezávislé (tj. obecné, generické) ontologie, které jdou na velkou úroveň podrobnosti a věnují se poměrně konkrétním pojmům, jako např. Cyc; naopak základní ontologie se soustředí na pojmy na vysoké úrovni abstrakce a mívají tudíž i nepříliš velký rozsah.

Již zmíněné ontologické závazky (ontological commitments) jsou základními teoretickými předpoklady týkajícími se přístupu k modelování reality, z nichž daná ontologie vychází, a které mohou být explicitně či implicitně vyjádřeny (někdy se pro souhrn ontologických závazků používá i výraz „cognitive bias"). Pokud používáme určitou ontologii, měli bychom si být vědomi, jaké ontologické závazky s sebou nese, protože to zároveň omezuje možnou integraci s ontologiemi nesoucími jiné ontologické závazky. V následující části vyjdeme z analýz uedených v pracích (Masolo, 2002a), (Masolo, 2002b), (Romanelli, 2006) a stručně popíšeme základní typy ontologických závazků.

Ontologie se může zahrnovat jednotliviny, obecniny nebo obojí. Jednotliviny jsou entity, které nemohou mít instance, zatímco obecniny je mít mohou (ač zrovna v daný moment žádnou mít nemusí).

Ontologie může být buď deskriptivní, nebo revizionistická. Deskriptivní ontologie vychází z běžného chápání skutečnosti („common sense"), bývá založena na vztazích vyplývajících z běžného jazyka a obvyklého porozumění, vývoj ontologie pak vychází především ze studia jazyka. Revizionistická ontologie se na přirozený jazyk a běžné porozumění neváže, snaží se vycházet z vědeckých a odborných přístupů a v případě, že jsou v rozporu s běžným rozuměním, dává jim přednost (Strawson, 1959).

Ontologie můžeme také rozdělit na multiplikativní nebo redukcionistické. Redukcionistický přístup se snaží vycházet z malé skupiny základních konceptů a ostatní koncepty rekonstruovat z této malé základní skupiny primitivních prvků. Multiplikativní ontologie se snaží o co největší expresivitu i za cenu větší složitosti a většího počtu základních konceptů. Pro multiplikativní přístup je také typické, že připouští kolokalizaci entit (více entit se může vyskytovat na jedné časoprostorové lokaci).

[5] Expresivita ontologie se nemusí odvíjet od expresivity použitého jazyka, ale týká se obsahu.

Při modelování skutečnosti můžeme také volit buď posibilistický, nebo aktualistický přístup. Aktualistická ontologie popisuje pouze to, co aktuálně existuje. Posibilistická popisuje i to, co může existovat, tzv. possibilie.

Podobná volba je i s ohledem na čas – buď prezentický, nebo eternální přístup. Prezentický přístup modeluje jen to, co v daný okamžik aktuálně existuje. Eternální přístup modeluje vše, co existovalo v minulosti, existuje nyní, či bude existovat v budoucnosti. Toto vše to je považováno za existující.

S tím související je i volba 3D nebo 4D modelovacího přístupu, která má vliv na modelování změn. Při použití třírozměrného modelovacího přístupu jsou běžné předměty plně přítomny v každém časovém okamžiku a jsou proměnné, neboli v různých okamžicích mohou mít různé vlastnosti. Při použití čtyřrozměrného modelovacího přístupu jsou běžné předměty chápány jako „časoprostoroví červi", jsou jen částečně přítomny v každém časovém okamžiku, skládají se tedy z „fází", v rámci kterých mohou mít různé vlastnosti.

1.8 V praxi využívané kategorie ontologií a vztahy mezi nimi

Svět, který nás obklopuje, lze popsat ontologiemi na různých úrovních obecnosti. Může být tvořen fyzikálními objekty, jejichž vlastností je to, že zaujímají místo v prostoru a čase, mají hmotu a energii a mohou na sebe vzájemně působit. Můžeme se ale také zajímat o abstraktní pojmy, které tyto vlastnosti nemají. Většinou však řešíme úlohy, ve kterých mají tyto obecné pojmy konkrétní podobu, závislou na problémové oblasti. Například biologie se zabývá živými organismy, což jsou speciální případy fyzikálních objektů, humánní medicína se zabývá lidmi, a proto dále specializujeme to, co víme o živých organismech, na člověka, popř. na organismy, které mohou jeho zdraví ovlivnit. Genetika dále upřesňuje předmět svého zájmu na objekty, jako je buňka, DNA, RNA atd. Tyto pojmy lze vymezit například pomocí již zmíněné aristotelovské formy definice, vycházející z hyperonyma (genus) a upřesňující podmínky (differentiae). Uvedenému rozvrstvení znalostí podle jejich obecnosti odpovídají také „úrovně" ontologií. Pojmy nezávislé na konkrétní oblasti bývají definovány v ontologii, která se obvykle označuje jako základní ontologie, popř. též vrchní ontologie nebo ontologie vyšší úrovně (v angličtině se používají výrazy „upper ontology", „top-level ontology", „foundational ontology", „fundamental ontology" apod.), a poskytují tedy primární rozlišení veškeré jsoucí skutečnosti do základních kategorií. Základní ontologie tedy zahrnují pojmy jako „věc", „fyzikální objekt", „čas", „abstraktní objekt", „agent", „oblast" atd.

Základní ontologie mají obecně tyto charakteristiky (Romanelli, 2006):

1. Poskytují referenci pro srovnávání různých ontologických přístupů a rámec pro analýzu, harmonizaci a integraci ontologií a metadatových standardů.

2. Poskytují výchozí bod pro tvorbu nových ontologií. Základní ontologie obsahují předdefinovanou skupinu ontologických entit, které mohou být znovu využity. V ideálním případě také základní ontologie definuje návrhové vzory pro běžně se vyskytující případy modelování.

3. Pomáhají uživateli vypořádat se s typickými problémy, na které může při návrhu ontologií narazit.

Ontologie popisující pojmy a vztahy v nějaké problémové oblasti se obvykle nazývají doménové ontologie. Doménové ontologie, které konceptualizují vybrané problémové oblasti se mohou odvolávat na pojmy ze základní nebo jádrové ontologie, protože tyto pojmy jsou jejich zobecněním. Doménové ontologie mohou být dále rozděleny podle úrovně specializace znalostí, viz uvedený příklad biologie, medicíny a molekulární genetiky. Rozdělení na jednotlivé vrstvy je ale relativní, protože závisí na předmětu zkoumání. Filosof bude pokládat za doménovou ontologii to, co pokládá biolog, fyzik, lékař a genetik za základní ontologii.

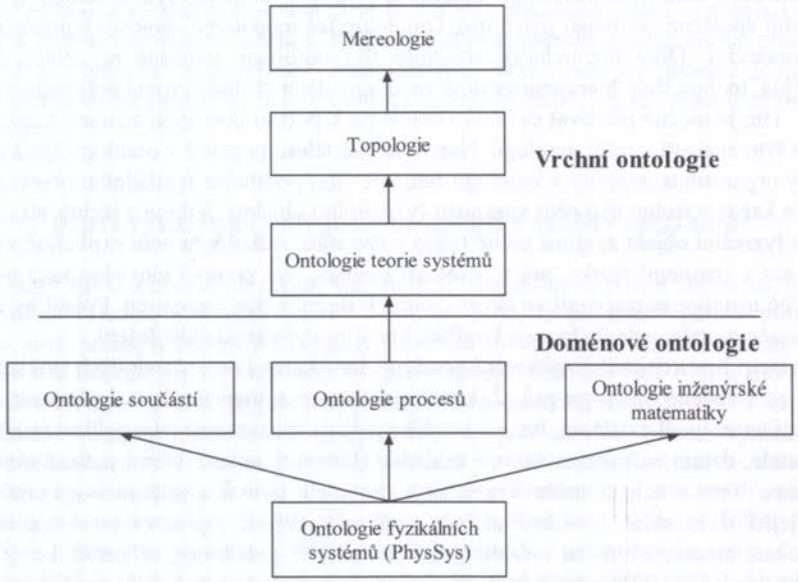
Základní ontologie usnadňují integrování různých doménových ontologií tím, že popisují společné vlastnosti jejich tříd. Ontologie lze modulárně spojovat a organizovat do hierarchií. Díky hierarchické struktuře tříd ontologií založené na relaci třída–podtřída, mohou třídy hierarchicky nižší ontologie dědit atributy ze tříd ontologie nadřazené. Tím je možné používat existující ontologie k popisu obecných znalostí a specializovat tyto znalosti v nižší ontologii. Například jak těleso popsané v ontologii fyziky, tak i živý organismus popsaný v ontologii biologie jsou podtřídou fyzikálního objektu. Instance každé z těchto tříd dědí vlastnosti fyzikálního objektu. Jednou z těchto vlastností je, že fyzikální objekt zaujímá určité místo v prostoru. Ačkoliv to není explicitně vyjádřeno ani v ontologii fyziky, ani v ontologii biologie, lze pomocí této vlastnosti ověřit, zda obě instance nezaujímají ve stejný okamžik stejné místo v prostoru. Pokud by taková situace nastala, jednalo by se o konflikt, který by vyžadoval další řešení.

Princip hierarchického spojování ontologií lze ukázat i na jednodušších příkladech, např. na uvedené ontologii psů. Zakončíme-li tuto ontologii na úrovni plemene, tedy třídami teriér, pudl a dalšími, lze pro každého jedince zaznamenat jeho pohlaví, majitele, chovatele, datum narození a navíc i základní vlastnosti sdílené všemi jedinci stejného plemene. Tuto ontologii může využít klub chovatelů teriérů a připojit svoji ontologii rozvíjející třídu teriér. Všechny atributy popisující obecně psy nová ontologie zdědí. Navíc tato ontologie umožní vytvořit podtřídy foxteriér, erdelteriér, velšteriér, kerry blue teriér a další. Tyto třídy připojí k již dříve uvedeným atributům psů další, např. zdravotní nebo genetické, dispozice, které jsou charakteristické pouze pro teriéry. Jinou ontologii vycházející z obecné ontologie psů mohou vytvořit chovatelé pudlů. Naopak, ontologie psů by mohla být hierarchicky podřízená ontologii savců a dědit odtud ty vlastnosti psů, které jsou společné všem savcům.

Příklad použití hierarchického spojení několika ontologií v komplexních průmyslových aplikacích uvádějí Borst a Akkermans (1997). Hierarchická struktura ontologií PhysSys, kterou tito autoři vyvinuli, je znázorněna na obr. 1.4.

Cílem ontologie PhysSys je poskytnout znalostní podporu pro vytváření inženýrských modelů v oblastech mechatroniky a termodynamiky. Obecné vlastnosti popisují tři hierarchicky uspořádané základní ontologie. Mereologická ontologie zavádí pojmy, které umožňují popsat jednotlivé subsystémy a jejich části až na úroveň jednotlivých součástí. Topologická ontologie poskytuje prostředky k popisu různých konfigurací objektů definovaných v mereologické ontologii. Ontologie teorie systémů využívá možné konfigurace subsystémů a součástí k zavedení pojmů známých z teorie systémů, jako je systém, jeho hranice, prostředí, otevřený nebo uzavřený systém apod. Tyto ontologie poskytují „jazyk" a základní znalosti, který používají tři doménové ontologie. Ontologie součástí

popisuje fyzikální systémy, subsystémy, součásti a jejich konfigurace. Chování těchto systémů, např. elektrické proudy nebo toky tepelných energií, jsou vyjádřeny pomocí ontologie procesů. Ontologie inženýrské matematiky potom dodává příslušné formální prostředky pro zpracování. Všechny tři doménové ontologie nakonec využívá ontologie PhysSys, která zabezpečuje vzájemné mapování odpovídajících pojmů. Při implementaci se autoři řídili základním metodickým doporučením: pokud lze, využijte existující ontologie. Tři základní ontologie byly z velké části převzaty z jiných zdrojů a integrovány do PhysSys.



Obr. 1.4 Hierarchie ontologií – struktura ontologie PhysSys.

1.9 Mapování ontologií, základní ontologie a sémantické lexikony

Jak jsme ukázali, řešení složitějších odborných úloh obvykle vyžaduje znalosti z více než jedné problémové domény. Dalším příkladem může být cestovní kancelář, která nabízí svým zákazníkům dovolenou, poskytuje komplexní službu zahrnující dopravu do místa dovolené, ubytování, pojištění, stravování a popřípadě zábavu. Mnoho podobných úloh lze najít v oblasti e-commerce a e-governmentu. Jejich společným rysem je to, že výsledek je složen, podobně jako v případě inženýrských ontologií, z mnoha dílčích řešení. Jednotlivé úlohy jsou většinou delegované různým agentům, mezi nimiž je třeba zajistit komunikaci tak, aby byl celkový výsledek konzistentní a splňoval zadání. Lze očekávat, že ontologie podporující dílčí úlohy jsou sestaveny nezávisle, a proto mohou vznikat různé konfliktní situace. V nejjednodušším případě se bude jednat o použití

různých výrazů pro stejný význam, tedy o synonymii. Osoba, která si koupí od cestovní kanceláře zájezd, se nazývá pro cestovní kancelář „zákazník", pro dopravce „cestující", pro hotel „host", pro pojišťovnu „pojištěnec", pro zábavní instituci „návštěvník". Přitom jde stále o stejného člověka. Pojem „dítě" může mít pro různé služby různou definici: pro dopravce, hotel nebo zábavní instituci mohou být různé věkové limity pro poskytnutí slev. Obecně lze očekávat, že v úlohách e-commerce a e-governmentu mohou být vztahy mezi pojmy ještě složitější.

Existují v podstatě dva přístupy, jak kombinovat znalosti z několika rozdílných, ale vzájemně souvisejících domén (o automatických metodách mapování ontologií se zmíníme podrobněji v kapitole 2):

1. Spojit dílčí ontologie do jedné a odstranit nekonzistence. V dalším řešení se bude využívat tato jediná výsledná ontologie. Obtížná úloha sémantického mapování je řešena pouze jednou, a to na začátku. Musí se však opakovat při každé změně libovolné dílčí ontologie.

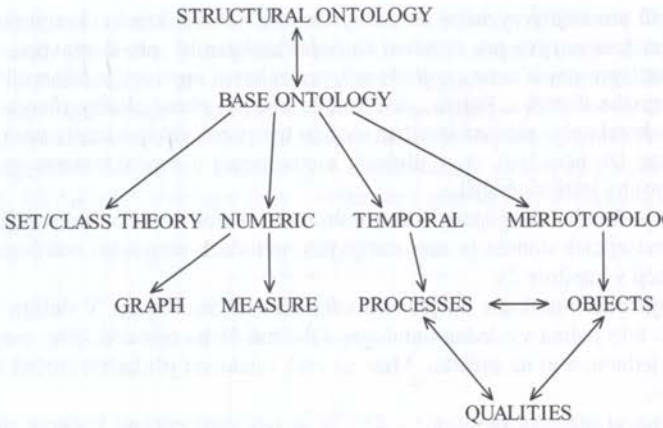
2. Zachovat původní ontologie s tím, že se nekonzistencemi budeme zabývat až v průběhu řešení konkrétní úlohy. Složitější algoritmus sémantického mapování pojmů mezi ontologiemi se bude řešit pokaždé, když se problém nekonzistence vyskytne. Výhodou ale může být, že se v konkrétní úloze na obtížné případy nenarazí a navíc změny dílčích ontologií se mohou dělat nezávisle.

V obou případech je výhodné, pokud najdeme obecnější významy, které spojují pojmy z jednotlivých heterogenních ontologií. Takovýto společný předek totiž dovoluje usuzovat o vlastnostech svých potomků a může přispět k odstranění nejednoznačnosti a propojení jednotlivých domén. V hierarchii ontologií je základní ontologie nejvýš, a tak umožňuje nalézat společné třídy (hyperonyma) k pojmům z různých ontologií. Návrh doménových ontologií většinou nezačíná vytvořením základní ontologie a postupným konkretizováním pojmů. Je obvyklejší, že se nejprve vytvoří (nebo ještě lépe identifikují existující) doménové ontologie na požadované úrovni obecnosti, jejichž pojmy se zpětně napojí na některou existující základní ontologii.

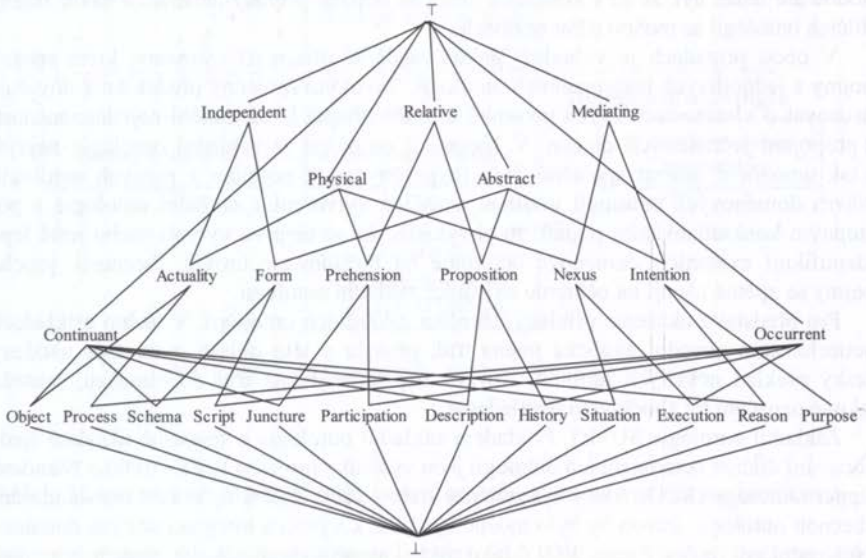
Pro představu ukážeme příklady několika základních ontologií. V těchto příkladech ponecháváme původní anglická jména tříd, protože v této oblasti neexistuje ustálený český překlad některých termínů. Rovněž nepíšeme jména tříd do rámečků, protože takové označení by zhoršovalo přehlednost.

Základní ontologie SUMO. Příkladem základní ontologie a současně ukázkou sjednocování dílčích heterogenních ontologií jsou výsledky projektu IEEE P1600.1 Standard Upper Ontology (SUO), který byl zahájen v roce 2000. Cílem bylo vytvořit standardní obecnou ontologii, kterou by bylo možné využívat k vývoji a integraci nových doménových ontologií (Niles, Pease, 2001). Jako základ autoři vybrali několik dostupných ontologií a vyvinuli novou, integrovanou ontologii nazvanou Standard Upper Merged Ontology (SUMO). Ontologie, které byly pro vývoj SUMO použity, byly vhodně strukturované, formálně popsané, dostatečně obecné a musely být „open source". Výsledná struktura ontologie SUMO je ukázána na obr 1.5.

SUMO vychází z jiných základních ontologií, např. z ontologie, kterou vyvinul J. Sowa (2000), ze základní ontologie popsané Russellem a Norvigem v práci (Russell, Norvig, 1994), z Allenových axiomů pro reprezentaci času (Allen, 1984) a z dalších zdrojů. Některé z nich jsou ukázány v následujícím textu.



Obr. 1.5 Ontologie SUMO a její komponenty.



Obr. 1.6 Obecná ontologie dle Sowy.

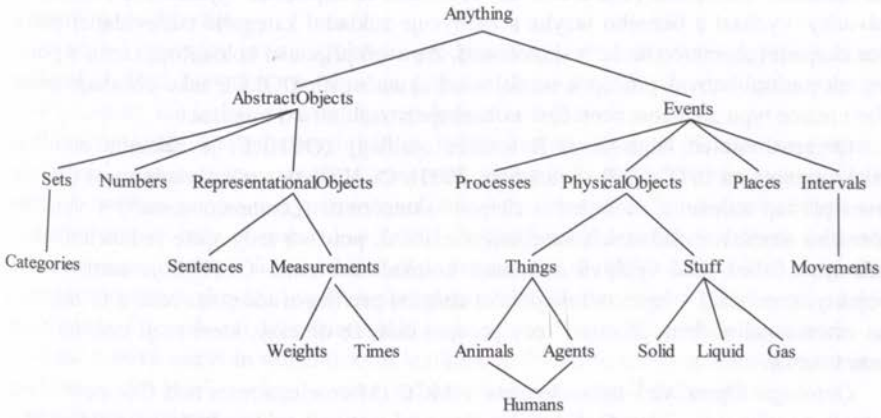
V základní ontologii dle Sowy (obr. 1.6) se univerzální třída označuje symbolem ⊤ a nazývá se Entity. Všechny ostatní třídy jsou podtřídami Entity. Díky dědičnosti, jsou také všechny instance instancemi Entity. Na opačném konci ontologie je symbol ⊥, nazvaný Absurdity. Žádný objekt není instancí třídy Absurdity a každá další třída je její nadtřídou. Některé třídy jsou popsány výrazem, který má gramatický tvar přídavného jména označujícího vlastnost (např. Physical). V tom případě tento výraz chápeme jako

vlastnost pojmu entity, tedy např. Physical Entity. Všechny třídy, které jsou bezprostřední podtřídou třídy T, mají přesně definovanou sémantiku pomocí logických výrazů. Například objekty třídy Physical jsou vymezeny podmínkou, že mají prostorové souřadnice a existují v nějakém čase, třída Continuant je charakterizována podmínkou, která je nezávislá na čase, zatímco třída Occurrent je popsána podmínkou závislou na čase. Třídy Entity a Absurdity si připomeneme v diskusi o reprezentaci ontologií výrazy jazyka OWL. Odpovídajícími třídami OWL jsou owl:Thing a owl:Nothing .

Definiční podmínky podtříd jsou logickou konjunkcí jejich rodičů, tedy například pro třídu Form platí Form = Independent ∩ Abstract. Jednotlivé třídy lze dále rozvíjet, což ale na obr. 1.6 pro jeho přehlednost neukazujeme. Například třída Object se rozpadá do několika podtříd v závislosti na dalším rozdělení vlastností, jež jsou vyjádřeny pomocí přidaných axiomů. Podrobné vysvětlení této ontologie lze nalézt v publikaci (Sowa, 2000).

Základní ontologie podle Russella a Norviga má téměř stromovou strukturu (obr. 1.7).

Nejobecnější třída v Russellově a Norvigově ontologii, která odpovídá třídě Entity v ontologii Johna Sowy, se zde jmenuje Anything. Uspořádání tříd je intuitivní, a proto nevyžaduje podrobnější diskusi. Vzhledem k relativně malému rozsahu uvedených dílčích ontologií autoři SUMO postupně sjednocovali třídy a jejich definice a případné konflikty řešili případ od případu. Obě uvedené ontologie byly reprezentovány formálně pomocí jazyka predikátové logiky, což také usnadnilo jejich sjednocování.



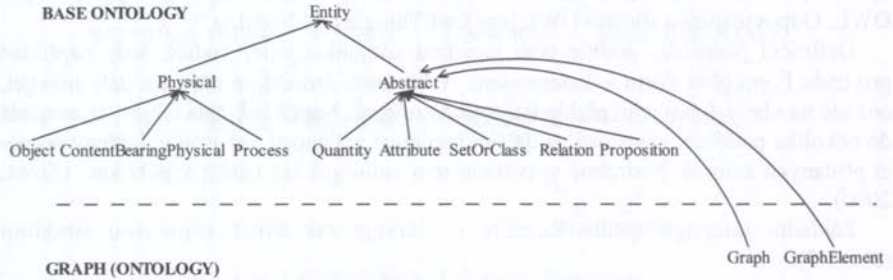
Obr. 1.7 Základní ontologie dle Russella a Norviga.

Několik nejvyšších tříd výsledné BASE ONTOLOGY (obr. 1.6) spolu se dvěma třídami ontologie GRAPH, která je taky součástí SUMO, je ukázáno na obr. 1.8.

Obrázek ukazuje, jak jsou třídy Graph a GraphElement, které patří do ontologie GRAPH, namapovány jako podtřídy třídy Abstract ontologie BASE ONTOLOGY.

Výsledná ontologie SUMO (Pease, 2006) obsahuje přibližně 1000 tříd a několik tisíc logických formulí, které je popisují.

Basic Formal Ontology (BFO)[6] byla vyvinuta týmem B. Smithe na Univerzitě v Lipsku (institut IFOMIS). BFO sestává ze série subontologií, které reprezentují určité perspektivy pohledu na realitu. Nejdůležitější jsou přitom SNAP, což je série ontologií, kdy každá představuje katalog entit existujících v daném časovém okamžiku, a SPAN, což je katalog procesů probíhajících napříč časem.



Obr. 1.8 Část sjednocené BASE ONTOLOGY.

Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE) je základní ontologie vyvinutá v institutu ISTC-CNR v Římě, původně jako referenční ontologický modul pro knihovnu ontologií v rámci projektu WonderWeb.[7] DOLCE je založena na metodice OntoClean (Oltramari, 2002) a snaží se explicitně vyjádřit své ontologické závazky: vychází z běžného jazyka a zachycuje základní kategorie odpovídající běžnému chápání („common sense") skutečnosti. Zároveň připouští kolokalizaci entit a používá tak multiplikativní přístup k modelování skutečnosti. DOLCE také obsahuje primitivní relace typu závislost nebo část a obsahuje rozsáhlou axiomatizaci.

Object-Centered High-Level Reference ontology (OCHRE) je základní ontologie také vyvinutá na ISTC-CNR (Schneider, 2003). OCHRE se snaží zkombinovat deskriptivní přístup založený na běžném chápání skutečnosti („common sense") s využitím menšího množství základních kategorií a axiomů, používá tedy spíše redukcionistický přístup, z čehož také vyplývá odmítnutí kolokalizace entit. OCHRE je zaměřena na objekty, které mají v jejím ontologickém chápání privilegované postavení, a je založena na extensionalistickém přístupu, tedy předpokládá, že objekty, které mají totožné části, jsou totožné.

Ontologie OpenCyc[8] byla vyvinuta v MCC (Microelectronics and Computer Consortium) v rámci projektu Cyc, který měl modelovat velkou část běžného chápání skutečnosti. OpenCyc je tou základní částí výsledné ontologie, která je volně dostupná. Celá ontologie Cyc pak pokrývá mnoho oborů i konceptů každodenního použití, obsahuje tisíce konceptů. Skupiny konceptů tvoří mikroteorie (kontexty), které se zabývají určitou vymezenou doménou.

[6] http://www.ifomis.uni-saarland.de/bfo/
[7] http://www.loa-cnr.it/DOLCE.html
[8] http://www.opencyc.org/

1.9.1 WordNet

V polovině osmdesátých let byl na Princeton University zahájen projekt, který trvá i v době vzniku této kapitoly. Jeho cílem je vytvořit úplný sémantický lexikon anglických slov a jednoduchých výrazů. Výsledkem je sémanticky provázaná databáze – sémantický lexikon – WordNet.

V osmdesátých letech používaly programy na zpracování přirozeného jazyka většinou jen omezený slovník. Na rozdíl od nich měl být WordNet svým rozsahem srovnatelný se slovní zásobou běžně používanou lidmi. Pro ilustraci uveďme, že obvyklá aktivní slovní zásoba člověka obsahuje 3000 až 10 000 slov, pasivní slovní zásoba je 3 až 6krát větší (Svobodová a kol., 2003). Uvádí se, že literární autoři s výjimečně bohatou slovní zásobou byli William Shakespeare a Karel Čapek, kteří používali až 40 000 slov. Oxford English Dictionary, slovník vydaný v roce 1989, obsahuje 231 100 záznamů (slov a frází). Tento slovník má 20 svazků, které mají dohromady 21 730 stránek (a váží 62 kg). Současná (rok 2008) verze WordNetu implementovaná pomocí databáze MySQL a označená jako verze 3.0 obsahuje 147 306 slov a frází.

Princetonský projekt vychází ze tří hypotéz (Miller, 1998a):

1. Předpokládá, že lze lexikální složku jazyka oddělit a studovat nezávisle od jeho ostatních forem (gramatiky, fonologie).

2. Dále předpokládá, že porozumění a efektivní využívání lexikálních znalostí je podmíněno existencí struktur a vztahů mezi lexikálními objekty.

3. Třetí hypotézou je již zmíněný požadavek na rozsah lexikonu, srovnatelný se slovní výbavou člověka.

První dvě hypotézy jsou v souladu s předpokládanými vlastnostmi glosářů, tezaurů a ontologií. Třetí hypotéza nevyhovuje požadavkům kladeným na ontologie, protože nejen připouští, ale i úmyslně zahrnuje mnohoznačnost jazyka.

WordNet používá terminologii (Miller, 1998b), kterou částečně přebíráme a již jsme užili dříve v tomto textu: množina synonym se nazývá synset, množina obecnějších pojmů k danému pojmu je hyperonymum a množina speciálnějších pojmů je hyponymum. Synset lze tedy přirovnat k množině doporučených a nedoporučených termínů, hyponyma jsou analogií termínů NT a hyperonyma BT dle (ISO 2788-1986). Jde však pouze o podobnost, protože uvedená norma pro návrh tezaurů nedbá příliš na sémantickou čistotu. Tu však ani WordNet zcela nezaručuje. Proto nebyl nikdy považován za ontologii a bývá nazýván sémantickým lexikonem. Přesto je ale významným nástrojem sloužícím k vytváření ontologií a navíc je i často sám jako ontologie používán.

Podrobnější informaci lze najít na http://wordnet.princeton.edu. Původní myšlenka projektu WordNet motivovala několik podobných projektů v různých evropských jazycích (např. EuroWordNet). V komunitě tvůrců národních „wordnetů" má významné postavení Centrum zpracování přirozeného jazyka na Masarykově univerzitě v Brně[9], které vytvořilo českou verzi WordNetu a propojilo ji s jinými verzemi.

[9] http://nlp.fi.muni.cz

1.10 Metodiky pro návrh ontologií

V současné literatuře lze nalézt mnoho metodických návodů jak navrhovat ontologie (Methontology; Uschold, Gruninger; IDEF5; UPON; Noy, McGuinness; Guarino, ...), z nichž některé jsou popsány v dalším textu. Většina z nich se v základních rysech shoduje v tom, že chápe úlohu jako speciální případ softwarového inženýrství a využívá známých postupů z této oblasti. Kromě toho však existují zásady, které jsou specifické pro návrh ontologií.

Zhruba lze nalézt tři hlediska, která jsou v metodikách zastoupena:

1. *Softwarové inženýrství.* Metodika se zabývá životním cyklem návrhu a implementací ontologií a z toho vyplývá celková filosofie návrhu i jednotlivé kroky.

2. *Reprezentace znalostí*, která vystupuje ve dvou rolích. Protože jde o úlohu softwarového inženýrství, je součástí návrhu ontologie dokumentovat jednotlivé kroky tak, jak bývá zvykem při návrhu programů. Navíc má být ale výsledkem znalostní model. Proto musíme vytvořenou konceptualizaci reprezentovat tak, aby tento model mohl být využíván dostupnými odvozovacími mechanismy při řešení předpokládaných úloh. V prvním případě jde o reprezentace procesu návrhu ontologie, v druhém o reprezentaci znalostí o popisovaném světě.

3. *Konceptualizace vybrané části reality.* Tato část metodiky je podstatou celé úlohy, je jedinečná pro návrh ontologií, a proto jí bude věnována zvláštní pozornost.

Porovnáním jednotlivých metodik a studiem existujících ontologií si můžeme všimnout toho, v čem se shodují:

1. Všechny metodiky připouštějí, že neexistuje jediný správný způsob modelování znalostí, vždy existují alternativy. Při rozhodování mezi nimi záleží na předpokládaném použití ontologie. Jestliže je cílem úlohy organizovat objekty reálného světa tak, aby je bylo možné snadno nalézt podle jejich významů (příkladem je katalog obchodního domu), postačí většinou pouze taxonomická struktura tříd. Pokud ovšem chceme využít ontologii k formálnímu odvozování (například v úlohách plánování robota), je zapotřebí třídy doplnit axiomatickými definicemi, se kterými bude odvozovací algoritmus pracovat.

2. Vývoj ontologie je vždy chápán jako iterativní proces. Změny v průběhu návrhu jsou důsledkem toho, jak si autoři ujasňují zadání a konceptualizaci úlohy a jak hledají kompromisy v komunitě těch, kteří budou ontologii sdílet. Kromě toho se vyvíjí vlastní problémová oblast a musí tedy existovat možnost ontologii aktualizovat. Jednoduchým příkladem je rozšíření služeb obchodnho domu, které má za následek změny v katalogu zboží, složitějším případem je zahrnutí nejnovějších poznatků do genetické ontologie, např. Gene Ontology, viz http://www.geneontology.org.

3. Pojmy, které ontologie používá, by měly věrně modelovat objekty a vztahy v reálném světě. Termíny, které popisují objekty, jsou nejčastěji podstatná jména, vztahy jsou reprezentovány slovesy. V některých medicínských ontologiích lze například nalézt výraz „affected" znamenající „postižený" nebo „poškozený", což je forma přídavného jména (viz např. WordNet). Tento výraz má ale význam „postižený jedinec", jde tedy o jmennou frázi. Podobnou interpretaci pojmenování tříd jsme již uváděli při popisu obecné ontologie Johna Sowy.

Míra formalizace ontologií se často liší. Uschold a Gruninger (1996) rozlišují 4 úrovně reprezentace ontologií:
- neformální reprezentaci, např. popis přirozeným jazykem,
- semineformální reprezentaci, např. popis strukturovanou podmnožinou přirozeného jazyka,
- semiformální reprezentaci vyjádřenou například graficky v UML,
- formální reprezentaci, tj. reprezentaci s formální sémantikou, např. jazykem predikátové logiky 1. řádu, speciálními jazyky pro reprezentaci znalostí (např. KIF) nebo jazykem OWL (Web Ontology Language), který je zaměřen na reprezentaci ontologií v úlohách sémantického webu. O těchto jazycích bude zmínka v dalším textu.

Tyto úrovně však nemají ostré hranice. Kromě toho se v jednotlivých fázích návrhu uplatňují různě. Nejprve například charakterizujeme objekty a vztahy v problémové oblasti textem v přirozeném jazyce, a potom je postupně upřesňujeme a formalizujeme (viz iterativní charakter návrhu).

Podrobný popis a srovnání všech existujících metodik by příliš vybočovaly z rámce tohoto textu. Proto uvedeme pouze základní kroky, které lze nalézt ve většině metodik, zmíníme se o tom, jak je někteří autoři dále rozvíjejí a budeme se podrobněji věnovat pouze konceptualizaci.

1.10.1 Návrh ontologie podle L. Noyové a D. McGuinnessové

Metodika, kterou popsaly Natalya Noyová a Deborah McGuinnessová (2001) je tvořena těmito kroky:

1. Vymezením problémové oblasti, kterou má ontologie reprezentovat. Součástí tohoto kroku může být sepsání několika nejdůležitějších otázek, jejich zodpovězení by měla navrhovaná ontologie podporovat (Uschold, Gruninger, 1996). Tyto otázky jsou vyjádřeny neformálně, v přirozeném jazyce, a mohou později sloužit k vyhodnocení výsledné ontologie.

2. Posouzením možnosti využít některou z již existujících ontologií. Na internetu lze nalézt servery, které nabízejí ontologie zaměřené na nejrůznější témata a činnosti. Jen na serveru SourceForge se v roce 2009 nacházelo několik set ontologií, přičemž vysoce specializovanou ontologii Gene Ontology si již stáhlo přes 60 tisíc uživatelů. Výhodou je, že existující ontologie jsou dostupné v některém ze standardních jazyků pro reprezentaci znalostí a že jejich kvalita je často ověřena hodnověrným způsobem. Pokud je ovšem požadovaná problémová oblast reprezentována více ontologiemi, ale žádná z nich ji nepokrývá úplně, vzniká úloha srovnávání, mapování a spojování několika ontologií. Například by se mohlo stát, že řešíme nějakou úlohu z fyziky a pro daný problém nenalezneme jedinou vhodnou ontologii, ale existují dvě ontologie, jedna pro kinematiku a druhá pro dynamiku hmotného tělesa. Problém mapování a spojování ontologií většinou není jednoduchý (Euzenat, Shvaiko, 1998), protože jde o mapování na sémantické úrovni. V případě, že nenalezneme jednu nebo více ontologií, které by pokrývaly celý problém, lze buď nějakou částečnou ontologii doplnit a rozšířit, nebo použít dostupné chudší a nepřesnější znalostní struktury, jako jsou glosáře a tezaury, jako výchozí model při návrhu ontologie (viz diskuse ke glosářům v oddílu 1.5).

3. Sestavením seznamu výrazů, které popisují důležité pojmy v problémové oblasti. Jde o termíny, které mohou při řešení vystupovat v nějakém tvrzení, důkazu, nebo které slouží ke komunikaci s uživatelem. V této fázi není důležité, jak korespondují

s významy, nevadí tedy například, že v seznamu budou synonyma. Upřesnění vztahu výrazů a významů se zabývají další kroky metodiky.

4. Definicí tříd a vytvořením jejich hierarchie. Třídy jsou sémantické objekty, které mohou být reprezentovány několika různými výrazy. Ontologie je sémantickou konstrukcí, a proto musí přiřadit výrazy odpovídajícím významům a ty potom dále uspořádávat. Uschold a Gruninger (1996) doporučují vyjádřit definice základních pojmů neformálně přirozeným jazykem. Gómez-Perez s kolektivem (2004) navrhuje zahrnout do textového popisu i vztahy k ostatním pojmům v ontologii. Tím se usnadní pozdější vytváření formálních definic. Jako příklad uvádějí slovní popis třídy v ontologii dopravy a dopravních prostředků. Výrazy, kterými neformálně označujeme třídy a jejich vlastnosti (atributy), jsou psány kurzívou, přičemž názvy tříd začínají velkým písmenem, a názvy vlastností tříd, popř. jejich instancí, začínají malým písmenem. Takovou neformální definicí může být: „Třída Městská hromadná doprava je Hromadná doprava, ve které jsou vlastnosti výchozí stanice, cílová stanice a všechny zastávky ve stejné Lokalitě." Všimněme si, že v této definici již naznačujeme hierarchickou strukturu podle aristotelovského schématu genus–differentiae: Třída Městská hromadná doprava je definována pomocí nadtřídy Hromadná doprava a podmínky, jež vymezuje část hromadné dopravy, kterou budeme nazývat Městská hromadná doprava (je to ta, kde zastávky, včetně počáteční a cílové, jsou v jednom městě). Třída (tj. objekt, význam) Městská hromadná doprava může být také popsána alternativním výrazem MHD (synonymem). Oba tyto výrazy mohly současně existovat v seznamu, který jsme vytvořili dle bodu 3, ale odkazují k jedinému objektu. Podtřídou Městské hromadné dopravy může být třída Městská autobusová doprava a objektem, který patří do této podtřídy, je třeba linka 119. Obecnější třídou v hierarchii je například třída Služby obyvatelům. Uvedená hierarchie popisuje služby, ne dopravní prostředky. Třída Autobus proto není součástí této hierarchie, ale je zařazen do jiné hierarchie, která popisuje dopravní prostředky.

Při vytváření hierarchie lze postupovat zobecňováním od speciálních tříd k jejich nadtřídám, tedy zdola nahoru (bottom-up), nebo od obecných tříd specializací k podtřídám, tedy shora dolů (top-down). Objekty, které chybějí v seznamu dle bodu 3, přitom doplníme. V obou případech však může dojít ke zbytečným krokům. Při zobecňování můžeme opomenout třídu, která v původním seznamu nebyla, ale do ontologie patří. Nezahrnuli jsme ji, protože při návrhu hierarchie postupem zdola nahoru je obtížnější vyhledávat společné vlastnosti tříd a tedy využívat model genus–differentiae (Uschold, Grunninger, 1996). Při specializaci vyjdeme nejspíš z některé dostatečně obecné třídy (např. Objekt, Entita) tak, abychom zahrnuli všechny existující třídy. Můžeme však nejvyšší třídu zvolit příliš obecnou a postupnou specializací vytvářet zbytečné větve v hierarchii. Proto se doporučuje začít od klíčových pojmů a ty rozšiřovat oběma směry. Takováto strategie zprostředka, nazývaná „middle-out", je rozumným kompromisem a dává většinou upokojivé výsledky. Je navíc podpořena výsledky výzkumu v kognitivní psychologii, které naznačují, že pojmy „uprostřed" hierarchie poskytují nejinformativnější popis objektů.

5. Definicí dalších vlastností objektů a tříd – slotů. V předchozím kroku jsme předpokládali, že třídy jsou popsány intensionálně, to znamená pomocí vlastností, kterou musejí všechny objekty tříd splňovat (viz oddíl 1.2). Opakem by byla definice extensionální, tj. vyjmenování všech objektů tříd. Tou se ale zabývat nebudeme. Kromě definiční vlastnosti tříd, která současně dovoluje vytvářet hierarchii, mohou vstupovat objekty tříd

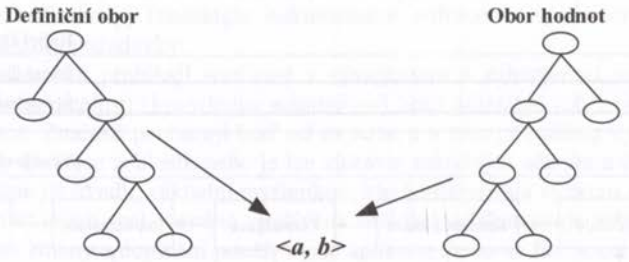
do relací s jinými objekty téže nebo jiné třídy. Většinou se omezujeme na binární relace, které označujeme jako atributy nebo též sloty třídy. Příkladem slotů třídy Hromadná doprava je výchozí stanice, cílová stanice a zastávky. Všechny tyto sloty musejí mít jako hodnotu jméno geografické lokality, tedy být objekty třídy Lokalita. Bývá zvykem, že sloty jsou pojmenovány slovesným výrazem, např. má_počáteční_stanici, má_cílovou_stanici, má_zastávky atd. Sloty popisují vlastnosti, kterými se jednotlivé objekty uvnitř třídy navzájem liší (linka 119 a linka 205 mají různé zastávky), zatímco vlastnosti, které definují třídu, musejí být totožné (obě linky patří městské hromadné dopravě). Každá podtřída dědí sloty všech nadtříd,

6. Definicí facet. Další vlastnosti slotů popisují facety. Nejčastěji určují některé z těchto vlastností slotů:

a) Nejmenší a největší počet hodnot, které může slot nabývat. Příkladem může být omezení, který říká, že minimální počet zastávek městské autobusové dopravy je 2, totiž výchozí a cílová stanice. Maximální počet zastávek nemusí být definován.

b) Typ hodnoty slotu. Hodnotou může být textový řetězec, číslo, logická konstanta True nebo False, popř. objekt nějaké třídy.

c) Definiční obor (domain) a obor hodnot (range). Definiční obor závisí na třídě, pro kterou je slot definován. Vzhledem k tomu, že je třída součástí hierarchie, existuje slot také ve všech podtřídách. Pokud je možnou hodnotou slotu objekt nějaké třídy, může být hodnotou také objekt libovolné podtřídy. Sloty vyjadřují binární relace, tedy dvojice $\langle a, b \rangle$, kde a je objekt z definičního oboru a b je objekt z oboru hodnot. Vzhledem k tomu, že jak definiční obor, tak i obor hodnot jsou součástí nějaké hierarchie, je zapotřebí určit jak „vysoko" v hierarchii je slot definován a jak vysoko je definována třída jeho funkčních hodnot. To je znázorněno na obr. 1.9.



Obr. 1.9 Definiční obor a obor hodnot slotu.

Doporučuje se zvolit definiční třídu i třídu oboru v hierarchii nejvýše, ale tak, aby slot měl ještě pro danou třídu význam a jeho možné hodnoty poskytovaly užitečnou informaci. Vzhledem k dědičnosti není třeba definovat slot pro podtřídy. Městská autobusová doprava může být součástí hierarchie, ve které může mít velmi obecné nadtřídy, např. Služba obyvatelům, která může být zobecněna na třídu Služba a dále na třídu Aktivita – viz Wordnet. Pro objekty třídy Služba obyvatelům může být užitečné definovat slot službu–poskytuje, avšak pro objekty třídy Aktivita by analogický slot neměl smysl.

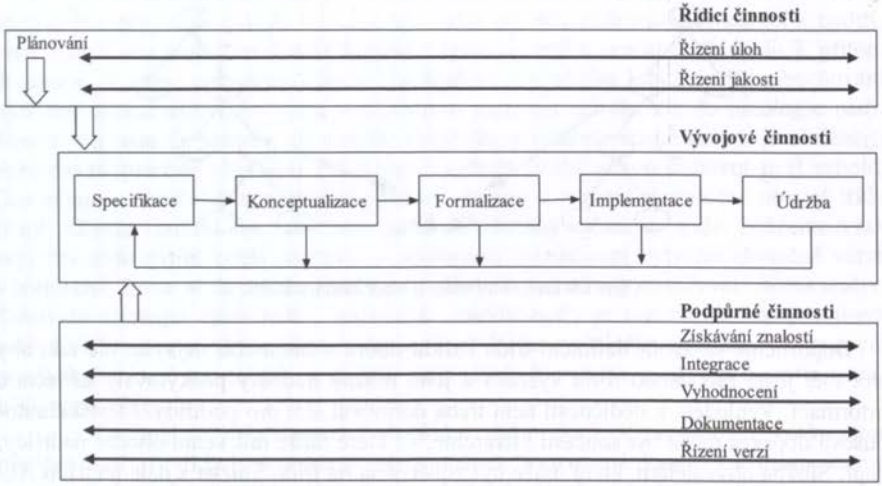
7. Vytvořením instancí tříd. Nejprve je třeba uvést, že instance tříd, které popisují jedinečné objekty problémové oblasti, patří obvykle do báze znalostí – popisují totiž obvykle aktuální stav světa, ale ne obecné vlastnosti konceptualizovaného světa. Báze znalostí se striktně řečeno nepokládá za součást ontologie. Jsou ale instance, které má smysl uvádět i v ontologii, pokud vyjadřují neměnné rysy světa vzhledem k dané konceptualizaci – příkladem jsou kontinenty a jejich jména v geografické ontologii. Někteří autoři však dokonce ontologie a báze znalostí neoddělují.

1.10.2 Metodika Methontology

Metodika pro návrh ontologií vyvinutá na Universidad Politécnica de Madrid (Gómez-Pérez a kol., 2004) se nazývá metodikou Methontology. Vychází z metod softwarového inženýrství a popisuje celý životní cyklus ontologie včetně úloh řízení vývojového procesu a verifikace výsledků. Základní filosofií je proces vyvíjejícího se prototypu (evolving prototype). Popis metodiky rozdělíme do dvou částí. Nejprve popíšeme strukturu a organizaci životního cyklu a potom se zaměříme na konceptualizaci, protože tato činnost odlišuje vývoj ontologií a od vývoje jiného softwaru.

1.10.2.1 Životní cyklus

Methonotology rozděluje proces návrhu ontologií do tří souběžně probíhajících skupin činností, které označuje jako vývojové, řídicí a podpůrné. Na obr. 1.10 jsou jednotlivé skupiny a jejich vzájemné závislosti znázorněny podle práce (Gómez-Pérez a kol., 2004). V nejdůležitější, vývojové skupině je ukázáno řazení jednotlivých činností. V obou zbývajících skupinách probíhají činnosti vždy od začátku až do konce cyklu, avšak jejich význam se v různých fázích vývoje se liší.



Obr. 1.10 Činnosti dle metodiky Methontology.

Základem návrhu je skupina vývojových činností. Prvním krokem vývoje je specifikace problému. V této fázi konstruktéři ontologie vymezí účel, proč má být ontologie vytvořena, jaké jsou předpokládané způsoby jejího použití a kdo bude cílovým uživatelem. Následuje nejdůležitější činnost, kterou je konceptualizace. Při ní se vytváří vlastní znalostní model popisované domény. Rozhodne se, které objekty a vztahy z domény budou v ontologii uvažovány a které lze ignorovat. Výsledný znalostní model může být zatím popsán neformálně, např. přirozeným jazykem. Dalším krokem vývoje ontologie je formalizace. K formálnímu vyjádření lze použít standardní prostředky, např. jazyk predikátové logiky, rámce nebo jazyky vyvinuté speciálně pro ontologické inženýrství, jako jsou KIF (Knowledge Interchange Format), Ontolingua, RDF, OWL a další.

Zvolená reprezentace má dosáhnout jednoznačného vymezení pojmů, nemusí však být výpočetním modelem, takže ji zatím nelze přímo zpracovávat počítačem. V následujícím kroku proto implementujeme formální reprezentaci ontologie ve vhodném programovacím prostředí. Posledním krokem vývojových činností je údržba vzniklého produktu. Jednotlivé činnosti jsme popsali jako posloupnost po sobě jdoucích kroků, ale z každé činnosti existuje možnost vrátit se zpět a opravit chybné řešení. Proto říkáme, že tento životní cyklus má charakter upřesňovaného, vyvíjeného prototypu, ale ne obvyklého vodopádového modelu.

Methontology definuje dvě další skupiny činností, jejichž hlavním cílem je vytvořit podmínky pro naplnění činností vývojových. Na obr. 1.10 jsou označeny jako řídicí a podpůrné. Ve skupině řídicích činností se uvádějí tři úlohy. První z nich, která je současně první úlohou v celé metodice, je vytvoření celkového plánu činností. Určí se, které konkrétní kroky budou pro vývoj potřeba, stanoví se jejich pořadí a přiřadí se jim potřebný čas a prostředky. Druhou, s tím související úlohou je vlastní řízení vývojových úloh. Cílem je zabezpečit, že jednotlivé vývojové činnosti začnou a jsou dokončeny tak, jak stanovil plán. Třetí úlohou této skupiny je řízení jakosti. Zde se průběžně ověřuje, že veškeré vytvořené výstupy (ontologie, dokumentace, softwarová implementace) splňují předepsané jakostní požadavky.

Podpůrné činnosti probíhají současně s vývojovými a zabezpečují jejich splnění. Metody získávání znalostí (knowledge acquisition) hrají důležitou roli zejména ve fázi konceptualizace. Znalosti pocházejí buď od experta, a v tom případě se využívají standardní metody elicitace znalostí, nebo je lze získávat strojovým učením z dat, nejčastěji z textu. Jak jsme již uvedli, základní myšlenkou, která motivovala výzkum a vývoj ontologií, je možnost jejich opakovaného využívání. Málokdy však existuje jediná ontologie, kterou lze beze změny a doplnění použít v jiné aplikaci. Proto ve fázi specifikace a konceptualizace často skládáme dohromady řešení z několika vzájemně se doplňujících ontologií a je tedy třeba řešit jejich vzájemné mapování a konzistentní spojování. Důležitou podpůrnou činností v metodice Methontology je proto integrace ontologií. Dříve, než budou ontologie používány v aplikacích, měly by projít testy ověřujícími, že splňují požadovaná kritéria. Verifikace prokazuje, zda ontologie splňuje požadavky specifikované během návrhu. Validace potvrzuje, zda definice v ontologii modelují vyhovujícím způsobem vybranou část reality. Při vyhodnocování výsledku návrhu se ověřuje, zda je ontologie konzistentní, úplná a úsporná – neobsahuje zbytečné části. Testy konzistence zkoumají, zda formální definice nedovolují odvodit spor. Úplnost ontologie znamená, že všechny podstatné znalosti z dané domény jsou buď v ontologii explicitně vyjádřeny, nebo je lze odvodit. Ontologie by měla být „stručná", obsahovat pouze potřebné definice

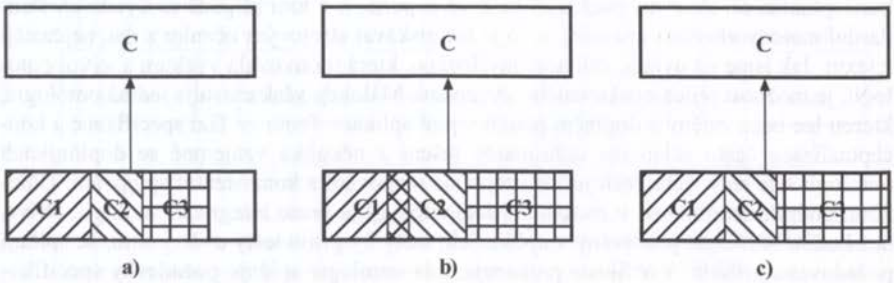
a vztahy, nemít redundantní definice ani neumožňovat jejich odvození. K validaci ontologií lze použít metody OntoClean, která bude podrobněji vysvětlena později. Dokumentace se týká všech fází a produktů vzniklých při návrhu. S tím souvisí i používání vhodného systému pro řízení verzí (content management), který umožní sledovat změny prováděné při vývoji. Vzhledem k tomu, že návrh je založen na myšlence vyvíjejícího se prototypu, je řízení verzí důležité při sledování vývoje.

1.10.2.2 Konceptualizace

Všechny části životního cyklu Methonotology slouží k podpoře nejdůležitější činnosti, kterou je konceptualizace. Konceptualizaci se také věnuje tato metodika podrobněji. Definuje jednotlivé dílčí úlohy a předpisuje pořadí, v jakém mají být prováděny. Jde o tyto úlohy a jejich pořadí:

1. Vytvoření glosáře termínů (výrazů), které budou v ontologii použity, včetně jejich synonym a akronymů. V ontologii budou tyto termíny popisovat třídy objektů, vztahy mezi třídami ontologie, vztahy k třídám jiných ontologií a některé jedinečné objekty (konstanty) v dané doméně. Interpretace každého termínu je také vysvětlena textem v přirozeném jazyce.

2. Vytvoření taxonomie tříd. Z glosáře vybere konstruktér ontologie termíny, které odpovídají obecným pojmům a budou reprezentovány jako třídy, a vytvoří z nich pomocí relace subsumpce taxonomii. Methonotology definuje tři různé varianty této relace: disjunktní dekompozici, úplnou dekompozici a rozdělení (partitioning). Disjunktní dekompozice třídy C je taková množina podtříd třídy C, ve které žádné dvě podtřídy nemají společnou instanci, ale přitom tyto podtřídy nepokrývají úplně třídu C. To znamená, že existuje instance třídy C, která není instancí žádné z podtříd. Úplnou dekompozicí třídy C je množina podtříd třídy C, které pokrývají úplně třídu C a mohou mít společné instance. Neexistují tedy instance, které nepatří alespoň do jedné podtřídy. Rozdělením (partitioning) třídy C je taková množina podtříd třídy C, ve které žádné dvě podtřídy nemají společnou instanci a pokrývají úplně třídu C. Protože ve všech případech platí, že instance libovolné podtřídy je současně instancí nadřazené třídy, jde vždy o relaci subsumpce. Jednotlivé případy jsou graficky znázorněny na obr. 1.11.



Obr. 1.11 Relace subsumpce: a) disjunktní dekompozice, b) úplná dekompozice, c) rozdělení.

3. Sestavení diagramu ad-hoc binárních relací. Cílem je definovat relace jak v rámci jedné taxonomie, tak i mezi taxonomiemi. Mezi dvěma třídami lze definovat více relací a ke každé relaci lze vytvořit inverzní relaci.

4. Vytvoření slovníku tříd. Tento slovník bude obsahovat všechny třídy, vlastnosti tříd a instancí, přiřazené relace a významné instance. Třídy jsou definovány intensionálně, tedy prostřednictvím vlastností (unárních relací), kterou musejí všechny instance splňovat. Navíc mohou existovat vlastnosti společné všem objektům třídy, které však nepokládáme za definiční. Takovéto vlastnosti v hierarchii tříd dědí jednak podtřídy, jednak jejich instance. Některé vlastnosti nemohou být spojené s třídou, ale musí být vlastnostmi instancí. Příkladem jsou vlastnosti, které umožňují rozlišit jednotlivé instance, např. její jméno nebo identifikátor. Podrobně se budeme zabývat těmito vlastnostmi při posuzování identity objektů v ontologii. Ve všech zmíněných případech však vždy jde o vlastnosti instancí třídy.

Kromě vlastností instancí existují také vlastnosti tříd. Jejich definičním oborem je třída a tyto vlastnosti nedědí ani podtřídy, ani instance. Příkladem vlastnosti třídy je jméno třídy, dokumentace nebo synonymum jména.

Instance tříd většinou nedefinujeme v ontologii, ale v bázi znalostí, která tuto ontologii užívá. Výjimkou jsou jedinečné instance, jež potřebujeme pro definici objektů. Do této kategorie také spadají konstanty potřebné v ontologii (např. největší a nejmenší uvažované reálné číslo). K třídám jsou přiřazeny relace, které je mají jako definiční obor.

5. Podrobný popis relací. Cílem je podrobně popsat všechny relace uvedené ve slovníku tříd. Popis obsahuje jméno relace, třídy, které jsou jejím definičním oborem a oborem hodnot, kardinalitu relace, inverzní relaci, pokud má smysl, a vlastnosti relace, jako je tranzitivita, symetrie atd.

6. Podrobný popis vlastností instancí. Ve slovníku tříd dle bodu 4 jsou u některých tříd uvedeny vlastnosti jejich instancí. Cílem této úlohy je detailně tyto vlastnosti definovat. Popis jistě obsahuje jméno vlastnosti instance, jméno třídy, ke které instance patří, obor hodnot a pokud existují i další charakteristiky. Těmi mohou být například minimální a maximální kardinalita, výchozí (default) hodnota, přesnost a rozsah hodnot a jednotky, ve kterých se hodnota vlastnosti instance měří. Dále lze uvést vlastnosti jiných tříd a instancí a vztah, ze kterých lze hledanou hodnotu vypočíst, a naopak vlastnosti jiných instancí a tříd, které lze pomocí hodnoty této vlastnosti určit.

7. Podrobný popis vlastností tříd a konstant. Tato úloha je podobná předchozí, pouze se týká vlastností tříd a konstant nebo jedinečných instancí.

8. Definice formálních axiomů a pravidel. Formální axiomy definují vztahy, které musí vždy platit a přitom je nelze přirozeně připojit k některému jinému objektu (třídě, instanci nebo relaci) ontologie. Používají se k ověření konzistence ontologie nebo báze znalostí. Pravidla slouží k odvozování hodnot vlastností instancí nebo tříd, popř. také k testování konzistence. Cílem této úlohy je detailně popsat formální axiomy a pravidla. Popis by měl obsahovat jméno axiomu nebo pravidla, slovní popis jako dokumentaci, formální vztah a dále třídy, vlastnosti, relace, konstanty a proměnné, které ve vztahu vystupují. Vztah bývá zapsán v jazyce predikátové logiky 1. řádu. Formální axiomy jsou uzavřené logické formule, pravidla mají tvar if ⟨předpoklad⟩ then ⟨důsledek⟩.

Některé ontologie jsou tvořeny pouze hierarchií tříd a souborem relací – viz 4 úrovně reprezentace ontologií dle Uschold a Gruningera (1996). V takovém případě lze Methonotology použít též, ale tento krok vynecháme.

Organizace FIPA (Foundation for Intelligent Physical Agents), která si klade za cíl podporovat rozvoj multiagentních systémů, doporučuje metodiku Methontology jako nástroj pro vytváření ontologií.

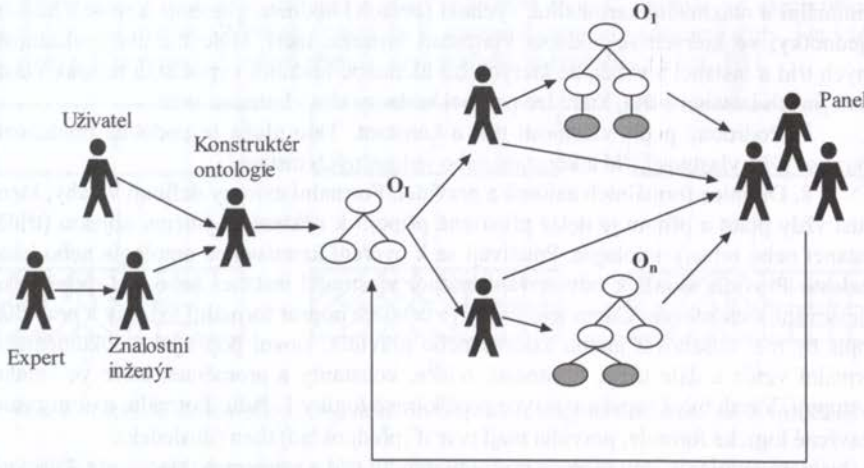
1.10.3 Metodika DILIGENT

Většina metodik pro návrh ontologií, včetně Methontology (Noy, McGuinness, 2001), (Uschold, Gruninger, 1996), vychází z centralizovaného vývojového modelu. V něm existuje vždy jediná ontologie, která se nachází v různém stadiu rozpracovanosti. Tuto ontologii upravují a vylepšují její tvůrci tak, aby splnila předpokládaná kritéria úplnosti, bezkonfliktnosti a úspornosti. Budoucí uživatelé se na těchto úpravách většinou nepodílejí.

Metodika DILIGENT (DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies), popsaná např. v práci (Vrandečić a kol., 2005), volí jiný přístup. Vychází z decentralizovaného modelu, ve kterém je sice základem jediná sdílená ontologie, ta je ale distribuována mezi větší počet uživatelů, kteří ji aktualizují a adaptují pro řešení svých úloh. Jejich výsledky potom vyhodnocuje zvláštní panel, který rozhoduje o tom, zda a jak se navrhované změny promítnou do sdílené ontologie.

Metodika sestává z 5 činností, které se cyklicky opakují:
- vytvoření sdílené ontologie,
- lokální adaptace,
- analýzy lokálních ontologií,
- aktualizace sdílené ontologie,
- lokální aktualizace.

Tento proces je znázorněn na obr. 1.12.



Obr. 1.12 Metodika Diligent podle práce (Vrandečić a kol. 2005).

Na počátku vytváří konstruktér ve spolupráci s expertem na danou problematiku první verzi sdílené ontologie O_1. Protože budoucí aktualizace sdílené ontologie jsou odvozeny z výsledků používání této ontologie v různých aplikacích, je vhodné, aby se někteří uživatelé podíleli i na tomto počátečním kroku. Na návrhu se dále může podílet také znalostní inženýr, který zprostředkovává přenos expertních znalostí ke konstruktérům ontologie. Výsledná ontologie není úplná, poskytuje pouze minimální jádro, na němž se shodnou všichni účastníci a které poskytuje dostatečnou podporu pro následující kroky.

Sdílenou ontologii adaptují uživatelé tak, aby podporovala jejich aplikace. Předpokládá se, že aplikace nejsou identické, i když mají mnoho společného. Výhodou tohoto uspořádání je, že vytváří různé pohledy a požadavky na konceptualizaci, které lze využít dále. Každý krok lokální adaptace musí být dokumentován a náležitě ospravedlněn.

Výsledné lokální ontologie spolu s vysvětlením a zdůvodněním všech změn a rozšíření vyhodnocuje panel složený ze zástupců všech profesí, které se podílejí na vývoji ontologie. Jsou v něm tedy uživatelé, znalostní inženýři, experti a konstruktéři ontologií. Panel analyzuje a rozhodne, které změny budou integrovány do sdílené ontologie. Předpokládá se, že výsledná architektura bude sestávat ze společné ontologie, která bude vyjadřovat sdílenou konceptualizaci domény a z ní vycházejících lokálních ontologií, jež budou popisovat jedinečné znalosti charakteristické pro každou aplikaci.

Na základě rozhodnutí panelu se integrují vybrané změny do sdílené ontologie, přičemž je třeba zajistit, aby jednotlivé změny nevytvářely konflikty, nekonzistence nebo redundance.

Aktualizovaná společná ontologie je východiskem pro další vývoj jednotlivých aplikací. Lokální adaptace, které rozšiřovaly a modifikovaly původní společnou ontologii, se musejí dát do souladu s její aktualizovanou verzí. Celý proces může být dočasně ukončen, nebo se zahájí nový cyklus.

Metodika DILIGENT byla navržena nedávno, ale přesto není nová. Poté, co ji její autoři formulovali jako nástroj pro vývoj ontologií, všimli si, že téměř shodný přístup používají biologové už dlouho. V roce 1735 navrhl Carl Linnaeus (Linné) ve své jedenáctistránkové publikaci Systema Naturae první taxonomii používanou v biologii. Ta byla první „sdílenou ontologií" tehdy neexistující metodiky DILIGENT, kterou začali používat biologové a zahájili cyklus dle obr. 1.12. V roce 1758, v desátém vydání Systema Naturae, již taxonomie obsahovala (dle Wikipedie) 4400 druhů zvířat a 7700 druhů rostlin.

Další významnou vlastností této metodiky je, že umožňuje přirozeným způsobem modelovat situaci, kdy je několik uživatelů nuceno spolupracovat, ale současně jsou si navzájem konkurenty. Spolupráce je nutí používat sdílenou část ontologie, lokální adaptace umožňuje skrýt část znalostí před konkurencí. Pro takovéto omezené sdílení znalostí se používá výraz „coopetition", který vznikl spojením slov „cooperation" a „competition". Význam omezeného sdílení znalostí v globálním tržním prostředí je zřejmý, a proto se mu v ekonomických vědách věnuje značná pozornost.

1.11 Jazyk, popis světů a reprezentace znalostí

V úvodu této kapitoly jsme popsali jednoduchý model jazyka, který sestával z množiny výrazů, množiny významů a zobrazení mezi těmito množinami. Řekli jsme, že synonyma jsou různé lexikální výrazy, které mají stejný význam. To znamená, že odkazují na stejný pojem. Příkladem mohou být dvojice lampa–svítilna, běžet–utíkat, nebo hezký–pěkný. Pro úvahy o glosářích nebo řízených slovnících (tezaurech) tento model postačuje. Aplikace předpokládají dobře vymezený kontext uvnitř problémové oblasti, a proto jsme ochotni obětovat přesnost popisu za zjednodušení vztahů mezi pojmy. Ukázali jsme například, že příslušná norma ISO připouští směšovat synonyma, pseudosynonyma, jména podtříd, instancí tříd a používat je jako alternativní názvy objektů. Pro anotování odborných textů takováto sémantická neurčitost nevadí. Ontologie jsou však určené k řešení podstatně složitějších úloh. Měly by definovat sémantiku vymezené části světa a formálně ji reprezentovat tak, abychom mohli z dat dále odvozovat nové poznatky. Chyby v reprezentaci vztahů mezi pojmy mají za následek chybné nebo neproveditelné inference. Proto musíme věnovat pozornost zásadám korektní konceptualizace a znovu se vrátit i k jazyku, který k tomu používáme. Srovnejme dvě dvojice výrazů: (1) lampa–svítilna a (2) manžel Blanky z Valois–zakladatel pražské univerzity. V prvním případě odkazují na pojem člověkem zhotoveného zdroje světla, ve druhém případě na osobu Karla IV. Lucemburského. Avšak zatímco v prvním případě nám jazykový cit nebrání považovat oba výrazy za synonyma, v druhém případě bychom výrazy jako synonyma nejspíš neoznačili. Druhý typ jazykových výrazů se nazývá obvykle jako určitá deskripce (definite description) a o jeho logické reprezentaci se vedou spory (Russell, 1905). Osoba Karla IV. je totiž popsána výrazy s různým smyslem, neboť by bylo možné, aby významem (denotátem) každého z nich byla jiná osoba. Prvním výrazem je Karel IV. popsán jako účastník partnerského vztahu, v druhém výrazu jako státník, tvůrce vzdělávacích a kulturních hodnot. Jsou tedy například myslitelné různé alternativní historie (možné světy), v nichž by manželem Blanky z Valois byly různé osoby. Výrazy vyjadřující význam a smysl zde používáme v duchu rozlišení zavedeného G. Fregem (1892), později přichází R. Carnap s návrhem používat mírně odlišné výrazy extense a intense. Podobný efekt můžeme často pozorovat, uvažujeme-li objekty v čase. Dvojice výrazů student–učitel, nebo ředitel nemocnice–lékař nebudeme pokládat za synonyma, přestože v konkrétních případech mohou odkazovat ke stejné osobě. Učitel byl kdysi studentem pedagogické školy. Lékař se mohl stát ředitelem nemocnice, tedy manažerem, o kterém předpokládáme, že nevykonává lékařskou praxi. Po nějaké době ale mohl ukončit řídicí práci a zase se vrátit ke svému původnímu oboru. Uvedené příklady naznačují, že existují běžné situace, které však neumíme vyjádřit pomocí dříve uvedeného modelu jazyka.

Popsané slabiny si logikové uvědomovali a hledali řešení. Carnap (1947) navrhl rozšířit model jazyka. Situace, které lze pomocí daného jazyka vyjádřit, charakterizuje souborem atomických vět. Pojem „atomická věta" chápe jako oznamovací větu tvořenou výrazem, který nazývá predikát, po němž následuje posloupnost výrazů, které označují objekty v popisovaném světě. Formálně budeme větu reprezentovat jako uzavřenou formuli jazyka predikátové logiky bez kvantifikátoů a budeme s ní zacházet jako s výrokem. Predikát vyjadřuje ve větě popisované vlastnosti objektů, objekty jsou reprezentovány individuovými konstantami jazyka. Pro rozlišení budeme psát predikáty s velkým

písmenem na začátku a individuové konstanty celé malými písmeny. Například výraz Ředitel(josef_novák) bude označovat větu „Ředitelem nemocnice je Josef Novák", Lékař(josef_novák) bude zastupovat větu „Josef Novák je praktikujícím lékařem", Člověk(josef_novák) reprezentuje „Josef Novák je člověkem" atd. Složené výroky lze konstruovat jako konjunkci, disjunkci nebo negaci jiných výroků s obvyklou sémantikou těchto operací. Pokud budeme uvažovat proměnné na místě objektů, lze vytvářet složené výrazy pomocí kvantifikátorů.

Atomická věta je pravdivá, pokud objekty reálného světa, ke kterým odkazují individuové konstanty, jsou ve vztahu, k němuž odkazuje predikát. Příkladem vlastnosti je zde „býti zakladatelem pražské univerzity" nebo „býti ředitelem nemocnice" a individuovou konstantou karel_4, nebo josef_novák. Pro jednoduchost jsme vyjádřili pomocí predikátů unární relace – vlastnosti objektů. Podobně bychom ale mohli vyjádřit i relace vyšší arity, např. Ředitel(josef_novák, nemocnice), pokud bychom chtěli explicitně vyjádřit, že Josef Novák je ředitelem nemocnice a ne třeba gymnázia.

Carnap nazval soubor vět, který obsahuje pro všechny atomické věty buď tuto větu samu, nebo její negaci, popisem stavu (state-description). Každý takový soubor popisuje jeden konkrétní logicky konzistentní stav našeho světa, konceptualizovaný pomocí zvoleného jazyka a je tím, co se v Kripkovské modální sémantice nazývá možným světem. Carnapův popis stavu tedy vyjadřuje možný, ne však nutně pravdivý popis reálného světa. Jeden z možných popisů stavu bude například říkat, že pražskou univerzitu založil Josef Novák, jiný bude vyjadřovat, že Karel IV. byl ředitelem nemocnice. Ani jeden z těchto popisů samozřejmě není pravdivý, protože příslušný vztah mezi individui neplatí, nebo neplatil v reálném světě. Kromě pravdivosti v reálném světě ještě Carnap zkoumá, zda je výrok splněn v popisu stavu. Atomický výrok (věta) je splněn v popisu stavu, pokud patří do popisu stavu. Složený výrok je splněn v popisu stavu, pokud ho lze dokázat pomocí operací konjunkce, disjunkce a negace aplikované na výroky, které jsou splněny v popisu stavu. Bude tedy existovat popis stavu, ve kterém jsou splněna tvrzení „Josef Novák založil pražskou univerzitu", jak jsme však uvedli, tento popis stavu ale není pravdivým popisem reálného světa.

Výroky, vyjádřené pravdivými atomickými větami, popisují jenom část našich znalostí o reálném světě. Další část se týká závislostí mezi atomickými výroky. V logice se výroky, které pokládáme za pravdivé, aniž bychom je logicky odvozovali, nazývají axiomy logické teorie. Jejich pravdivost vyplývá ze vztahů, které platí v popisovaném světě. Existují také axiomy, nazývané tautologie, které jsou pravdivé díky své logické struktuře, tedy nezávisle na vnějším světě. Ty zde ale neuvažujeme. Příkladem axiomu je výrok „Jestliže je Josef Novák lékařem, potom má vysokoškolské vzdělání." Tento axiom zapíšeme jako logickou implikaci

$$\text{Lékař(josef_novák)} \rightarrow \text{Má_VŠ_vzdělání(josef_novák)}.$$

Z pravdivostní tabulky implikace $A \rightarrow B$ víme, že hodnota implikace je vždy pravdivá s jedinou výjimkou, když A je pravda a B je nepravda. Tím, že přijmeme axiom Lékař(josef_novák)→Má_VŠ_vzdělání (josef_novák), říkáme, že tato implikace je pravdivá a že tedy nemůže nastat situace, kdy je Josef Novák lékařem, ale nemá vysokoškolské vzdělání. Ostatní tři možnosti Josefa Nováka, tedy (1) není lékařem – nemá vysokoškolské vzdělání, (2) není lékařem – má vysokoškolské vzdělání a (3) je lékařem – má vyso-

koškolské vzdělání, tento axiom připouští. O pravdivosti tohoto axiomu se nelze přesvědčit logickým důkazem z jiných výroků. Jeho platnost můžeme pouze ověřit na základě znalostí zákonitostí, které jsou vně logického systému. Pokud budeme tento axiom vztahovat pouze na osobu Josefa Nováka, tj. budeme ho chápat doslovně tak, jak je napsán, můžeme se o jeho pravdivosti v reálném světě přesvědčit tím, že si necháme předložit pracovní smlouvu a doklady o vzdělání Josefa Nováka. Pokud bychom chtěli tento axiom zobecnit a chápat ho jako univerzálně kvantifikovaný, tj. $\forall x$ (Lékař(x) → Má_VŠ_vzdělání (x)), ověříme jej studiem platných právních norem. Z hlediska reprezentace znalostí axiom vyjadřuje, že některé stavy pokládáme za nemožné. Ve všech možných světech, ve kterých budou platit právní normy vyjádřené tímto axiomem, z něj bude možné odvodit závěry, které jsou v souladu s těmito normami. Může ovšem také existovat možný svět, ve kterém lékař nemusí mít vysokoškolské vzdělání. V tomto světě logické důsledky uvedeného axiomu nebudou platit.

Ontologie nepopisují jedinečné, neopakovatelné případy, ale celou třídu možných světů, jejichž společné chápání – konceptualizaci – chce nějaká komunita sdílet. Tyto světy budou mít mnoho společného, ale budou se také lišit, protože budou například obsahovat různé instance popisovaných tříd. V jednom světě bude Josef Novák lékařem, v jiném bude ředitelem zdravotnického zařízení. Proto musíme zkoumat, které stavy jsme ochotni připustit jako možné, které stavy nutně musí nastat a které pokládáme za vyloučené. Ukážeme to na následujícím příkladu, v němž opět figuruje Josef Novák.

Pomocí tří predikátů Ředitel, Lékař a Člověk a jedné individuové konstanty josef_novák lze sestavit osm různých popisů stavu, ve kterých jsou jednotlivé výroky buď v pozitivním, nebo v negativním tvaru. Tři z těchto osmi popisů charakterizují světy, kterými se chceme zabývat. Jsou to:

1. Ředitel(josef_novák), ~Lékař(josef_novák), Člověk (josef_novák),
2. ~Ředitel(josef_novák), Lékař(josef_novák), Člověk (josef_novák),
3. ~Ředitel(josef_novák), ~Lékař(josef_novák), Člověk (josef_novák).

Vyloučili jsme 5 světů, které nepokládáme za možné z následujících důvodů:

1. Dle naší konceptualizace nikdo nemůže současně vykonávat povolání ředitele nemocnice a praktikujícícho lékaře.

2. V naší konceptualizaci je Josef Novák člověkem, a proto neuvažujeme světy, které obsahují ~Člověk (josef_novák).

Je zřejmé, že Josef Novák může být buď lékařem, nebo ředitelem, nebo nemusí být ani lékařem, ani ředitelem. Rozhodně ale víme, že je Josef Novák člověkem. Tyto vlastnosti vyjadřují následující definice.

1.11.1 Nutnost, možnost a metavlastnosti

Formálně můžeme nutnost a možnost vyjádřit takto:

Pokud je nějaká vlastnost P splněná ve všech uvažovaných světech, budeme říkat, že tato vlastnost nutně platí v těchto světech, což budeme značit $\Box P$. Pokud je vlastnost P splněná ve všech m o ž n ý c h světech, které lze z predikátů a konstant jazyka vytvořit, řekneme, že je logicky pravdivá.

Pokud je vlastnost P splněná alespoň v jednom uvažovaném světě, řekneme, že vlastnost P možná platí, což budeme zapisovat jako $\Diamond P$.

Platí, že $\Diamond P$ je zkrácený zápis pro $\sim\Box(\sim P)$. Vyjadřuje, že nemusí nutně platit vlastnost $\sim P$, tedy může platit vlastnost P. Podobně $\Box P$ je zkrácený zápis pro $\sim\Diamond(\sim P)$. Říká, že vlasnost P nutně platí, když není možné, aby neplatila.

V uvedeném příkladu platí □Člověk (josef_novák) a například ◊Ředitel(josef_novák). Formální definice, axiomatizace a podrobnější diskusi vlastností operátorů nutnosti a možnosti lze najít v publikacích, které se zabývají modální logikou, např. (Carnap, 1947), (Kripke, 1981), (Fagin a kol., 1995), (Ramsay, 1988).

1.12 Metodika pro analýzu ontologií – OntoClean

Zavedené pojmy nutnosti a možnosti jsou vlastnosti vlastností, tedy metavlastnosti. Protože v ontologiích definujeme pomocí společných vlastností objektů třídy a taxonomie tříd, můžeme analýzou vlastnosti tříd zkoumat, zda je konceptualizace správně utvořená. To je cílem a současně metodou řešení metodiky OntoClean, kterou vypracovali Guarino a Welty (2002). Na rozdíl od dříve uvedených metodik Noyové a McGuinnessové, Methonotology nebo DILIGENT, které jsou syntetické, protože poskytují návod jak sestavovat ontologie, je OntoClean metodikou analytickou, protože umožňuje posoudit, zda je existující ontologie správně utvořena. K tomu definuje OntoClean metavlastnosti rigidnost, identitu, jednotu a závislost a zkoumá, jak se tyto metavlastnosti promítají do správně sestavené ontologie.

1.12.1 Rigidnost

Existují vlastnosti, které daný objekt musí nutně mít v libovolném světě. Takové vlastnosti označujeme jako esenciální. Vlastnosti, které objekt pouze může mít, se nazývají neesenciální nebo také akcidentální (Stanford Encyclopedia of Philosophy, 2009). Esenciálnost je metavlastnost vyjadřující vztah mezi objektem, tj. instancí třídy, a vlastností, která třídu definuje. Pokud budeme jednotlivé popisy stavu indexovat časem, esenciální vlastnost platí v libovolném čase. Esenciální vlastností Josefa Nováka je to, že je člověkem, jeho akcidentální vlastností je to, že je lékařem nebo ředitelem, nebo má jiné nespecifikované povolání.

Welty a Guarino (2001) definují podobně pojem rigidní, nerigidní a antirigidní vlastnosti tříd takto:

1. Vlastnost P je rigidní, pokud je esenciální pro všechny objekty dané třídy, tedy pokud platí

$$\Box\,(\forall x, t\; P(x, t) \rightarrow \Box\forall t'\; P(x, t')).$$

2. Vlastnost P je nerigidní, pokud není esenciální pro některé objekty dané třídy, tedy pokud platí

$$\Diamond\,(\exists x, t\; P(x, t)\; \&\; \Diamond\forall t'\; \sim P(x, t')).$$

3. Vlastnost P je antirigidní, pokud není esenciální pro všechny objekty dané třídy, tedy pokud platí

$$\Box (\forall x, t\ P(x, t) \rightarrow \Diamond \exists t' \sim P(x, t')).$$

Třída „být člověkem" je rigidní. Všechny osoby patřící do této třídy musejí mít tuto vlastnost ve všech možných světech. Rigidní vlastnosti jsou označeny metavlastností $+R$, nerigidní jako $-R$ a antirigidní jako $\sim R$. Vlastnost „být člověkem" je tedy označena jako $+R$. Každá instance třídy, definované vlastností – každý člověk –, musí jí být po celou dobu své existence. Naproti tomu vlastnost „být ředitelem" je antirigidní, tj. $\sim R$. Každý člověk, který má mít tuto vlastnost, ji mít nemusí, může přestat být ředitelem.

Rigidnost, nerigidnost a antirigidnost jsou důležité metavlastnosti, které zásadním způsobem ovlivňují správné taxonomické uspořádání tříd. Jak víme, taxonomie je tvořena relací subsumpce, kterou jsme verbálně vyjadřovali pravidlem „všechny–některé". Formálně lze subsumpci zapsat vztahem $\forall x\ P(x) \rightarrow Q(x)$, který říká, že každý objekt (instance) třídy P je také objektem (instancí) třídy Q, třída P je tedy podtřídou třídy Q. Ve vztahu tříd a podtříd platí, že se antirigidní metavlastnost dědí. To znamená, že podtřídou třídy definované antirigindní vlastností nemůže být třída definovaná rigidní vlastností.

1.12.2 Identita

V úvodu jsme ukázali, že jednoduchý model jazyka umožňuje, aby dva výrazy odkazovaly ke stejnému objektu, a přesto je nepokládáme za synonyma. Abychom si však tyto problémy vůbec uvědomili, musíme být schopni rozhodnout, zda hovoříme stále o stejném jedinci. Otázka, zda je Karlel IV., manžel Blanky z Valois, stejná osoba jako Karel IV., jakožto zakladatel univerzity, se zdá být triviální, Lucemburk byl natolik prominentní osobou, že o tom nikdo nemá pochyby. Je však ředitel nemocnice stejný člověk jako osoba, kterou známe z dřívějška jako lékaře? Jak budeme tohoto jedince identifikovat? Otázka identity objektů, které chceme zahrnout do konceptualizace, je někdy složitá a řešení nemusí být jednoznačné. Objekty se mohou svojí podobou radikálně lišit, a přesto je můžeme pokládat za identické. Batole se svojí podobou liší od dospělého člověka, a přesto jde o stejného jedince. Objekty mohou být tvořené z identické fyzikální látky (ze stejných atomů), a přesto mohou být chápány jako různé. Sádrovou sochu pokládáme za jiný objekt než hromadu sádry, ze kterého ji sochař vytvořil. Hromada cihel je jiným objektem než hrubá stavba domu, přestože je tato hrubá stavba vytvořena (v zjednodušeném případě) pouze z těchto cihel. Pokud několik cihel z hromady odebereme, bude výsledkem jiná hromada, ale pokud odebereme stejný počet cihel z hrubé stavby, bude to stále táž stavba, příklady jsou citovány volně podle práce (Welty, Guarino, 2002). Můžeme se dále ptát: Zachovává si člověk svoji identitu, pokud dostane orgány někoho jiného? Podobnou otázkou je, zda se bude jednat o stejné auto, když u něj vyměníme ojeté pneumatiky za nové. A co když navíc vyměníme blatníky a zbytek karoserie? Je to stále totéž auto? A když nakonec vyměníme motor s převodovkou? Z původního auta zbyl pouze dokument, nazvaný Technický průkaz, ale i zde si můžeme požádat o duplikát! Je zřejmé, že ontologie musí používat ke konceptualizaci vlastnosti, které umožňují zařazovat objekty do tříd, současně ale musí mít prostředky k tomu, abychom různé objekty stejné třídy rozlišili. Musíme být tedy schopni rozhodnout, zda je moje auto, přestože má nové blatníky, stále pokládáno za to původní auto a není to

auto stejné značky patřící mému sousedovi. Je důležité, abychom si problém identity při návrhu ontologií uvědomili, přestože na něj často nemáme jednoduché odpovědi.

Identitou objektů reprezentovaných svými vlastnostmi se, kromě jiných, zabýval i Gottfried Wilhelm Leibniz, který navrhl toto kritérium:

> *Dva objekty A a B prohlásíme za identické, pokud objekt A má každou vlastnost, kterou má objekt B, a naopak objekt B má každou vlastnost, kterou má objekt A.*

Leibnizova definice vyvolala mnoho diskusí a dalších úvah. Především je zřejmé, že by nebyla příliš užitečná, pokud bychom v definici změnili směr implikace. Jsou-li totiž objekty A a B identické, jde o jediný objekt a podmínka shodnosti vlastností je triviálně splněná (tento princip se nazývá zákon stejnosti identických). Chápeme-li podmínku jako postačující, tj. tak, jak je v definici napsaná (tento princip se nazývá zákon identity stejných), je často obtížné prokázat její splnění pro všechny vlastnosti (Kolář, 1999). Definici lze ale využít v pozměněné a oslabené formě.

Pro objekty třídy lze definovat podmínku identity IC (identity condition), pomocí které budeme o identitě instancí třídy rozhodovat. Pokud objekty nesplňují tuto podmínku, jde o dva různé objekty, v opačném případě jde o objekt jediný. V případě identity lidí může být podmínkou identity stejný profil DNA, otisky prstů nebo třeba rodné číslo – záleží pouze na tom, kterou z těchto podmínek v daném kontextu preferujeme. Finanční úřad identifikuje lidi pomocí rodného čísla, pro identifikaci Karla IV. bychom však spíše použili profil DNA.

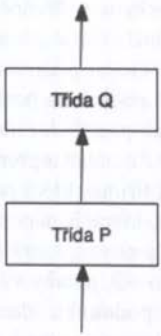
Metodika OntoClean dále rozlišuje, zda třída podmínku identity poskytuje (supplies), nebo jenom přenáší (carries). Třída poskytuje podmínku identity, pokud ji nezdědí ze třídy postavené v hierarchii výše. K tomu, aby třída mohla poskytovat podmínku identity, musí být vlastnost, která třídu definuje, rigidní a podmínka identity v ní musí být definována. Třída podmínku identity přenáší, pokud ji získá subsumpcí (zdědí ji) od hierarchicky vyšší třídy, která ji poskytuje. Předpokládáme, že pokud třída podmínky identity poskytuje, tak ji také přenáší. Definujeme-li podmínku identity v hierarchii Živočich–Člověk–Muž, můžeme definovat podmínku identity ve třídě Člověk pomocí rodného čísla. V tom případě je tato podmínka poskytována třídou Člověk a přenášená třídou Muž. Pokud bychom definovali podmínku identity ve třídě Živočich pomocí profilu DNA, tak je poskytována třídou Živočich a přenášená třídami Člověk a Muž. Metodika OntoClean zavádí pro vlastnost (třídu), která poskytuje podmínku identity, označení $+O$ (own identity), pro vlastnost, která ji neposkytuje, označení $-O$, pro vlastnost, která podmínku identity přenáší, $+I$ a pro tu, která ji nepřenáší, $-I$. Zjevně platí, že pokud je nějaká vlastnost označená jako $+O$, je taky $+I$ a $+R$.

Pokud nelze jednoduše stanovit podmínku identity, lze někdy přesto využít metavlastnosti k prokázání, že dva objekty jsou různé. Platí, že pokud je nějaká vlastnost objektu A pro něj esenciální, avšak stejná vlastnost není esenciální pro objekt B, tyto objekty nemohou být identické. Tímto krokem tedy nedokazujeme identitu, ale rozdílnost objektů. Například esenciální vlastností sádrové sochy je její tvar. Tvar však není esenciální vlastností hromady sádry. Proto nejsou sádrová socha a hromada sádry identickými objekty. Podobnou úvahu lze použít u posouzení, zda je hrubá stavba domu identická s hromadou cihel, ze kterých je vytvořena.

1.12.3 Jednota

S určováním identity objektů souvisí také otázka jednoty (unity) instancí tříd. Zatímco problémem identity je rozhodnout, zda jsou dvě instance rozdílné, nebo zda jde o jedinou instanci, cílem posouzení jednoty instance je určit, co je a co není její částí, jaké jsou její hranice a za jakých podmínek lze instanci chápat jako jeden celek. Řekneme, že objekt je celek vytvořený (definovaný) relací *R*, pokud všechny části nějakého rozdělení tohoto objektu jsou spojeny touto relací a žádná část tohoto objektu není touto relací spojena s něčím, co není součástí tohoto objektu. Například školní třída, ve smyslu skupina žáků, je celek vytvořený relací „míti společného třídního učitele", kámen je celek vytvořený fyzikálním spojením částí horniny, souhvězdí je celek vytvořený prostorovým uspořádáním hvězd. Ve všech případech jsme schopni definovat hranice celku a tak rozhodnout, co do celku patří a co ne. Kromě toho existují objekty, které nejsou celky. Jak jsme ukázali, existují různé typy relací, kterými lze definovat jednotu. Hovoříme o podmínce jednoty (unity condition UC). V případě kamene se jedná o topologickou relaci, souhvězdí je definované morfologickou relací a školní třída funkční relací.

Také třídy můžeme hodnotit podle toho, zda jsou nebo nejsou jejich instance celky. Pokud existuje pro všechny instance třídy stejná relace *R*, která definuje tyto instance jako celky, říkáme, že třída přenáší jednotu a značíme to $+U$. Příkladem takové třídy je „býti kamenem". Každý kámen, každá instance této třídy je celkem. Pokud všechny instance třídy mohou nebýt celkem, říkáme, že třída přenáší antijednotu, a označujeme to $\sim U$. Třída Voda, jejíž instancí jsou různá množství vody, přenáší antijednotu. Pokud existují pro všechny instance třídy různé relace R_i, které definuje tyto instance jako celky, říkáme, že třída nepřenáší jednotu, což značíme jako $-U$. Příkladem takové třídy je třída definovaná relací „býti právnickou osobou". Právnická osoba může být podnik, ve kterém je celek definován relací podřízený–ředitel, nebo družstvo, ve kterém je celek definován relací „člen družstva", tedy dvěma různými relacemi.



Obr. 1.13 Hierarchie tříd.

Metodika OntoClean (Guarino, Welty, 2002), (Welty, Guarino, 2001), (Goméz-Pérez a kol., 2004) používá metavlastnosti rigidnost, identitu a jednotu k posouzení toho, zda je taxonomická struktura správně utvořena. Základní myšlenka metodiky OntoClean vychází z toho, že subsumpce je relace, která jednak umožňuje dědění z třídy na

podtřídu, ale hlavně zaručuje, že instance podtřídy musejí být také instancemi všech rodičovských tříd. Z toho vyplývají omezení, která lze získat právě analýzou metavlastností tříd. Pro vztah dvou tříd dle obr. 1.13 platí těchto pět omezení:

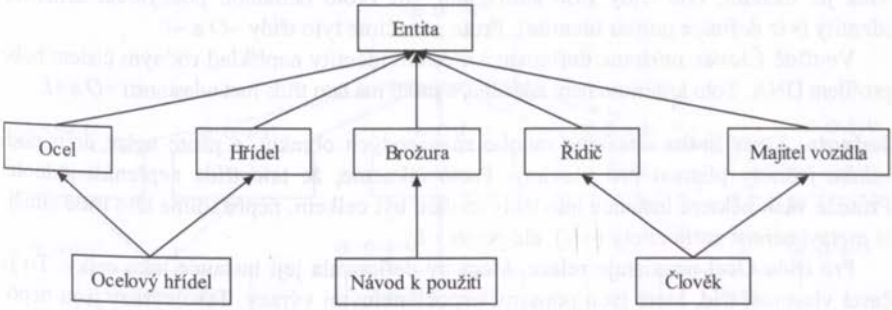
1. Pokud je třída Q antirigidní, potom musí být třída P antirigidní, $Q{\sim}R \rightarrow P{\sim}R$,
2. Pokud třída Q jen přenáší identitu, potom musí třída P přenášet identitu, $Q{+}I \rightarrow P{+}I$,
3. Pokud třída Q jen přenáší jednotu, potom musí třída P přenášet jednotu, $Q{+}U \rightarrow P{+}U$,
4. Pokud třída Q jen přenáší antijednotu, potom musí třída P přenášet antijednotu, $Q{\sim}U \rightarrow P{\sim}U$,
5. Třídy, které mají nekompatibilní podmínku identity (IC) nebo podmínku jednoty (UC), musejí být disjunktní.

V následujícím příkladu ukážeme, jak metodika OntoClean používá těchto omezení k ověření toho, zda je ontologie správně sestavena a v případě, že jsou některá omezení porušená, nabídne řešení.

1.12.4 Příklad použití metodiky OntoClean

Jednotlivé kroky metodiky OntoClean ukážeme na příkladu jednoduché ontologie dle obr. 1.14. Tato ontologie byla sestavena s cílem ukázat podstatu možných chyb.

Třídy, které budeme analyzovat, jsou uvedeny v rámečku, relace jsou znázorněny plnou čárou. Pokud není k čáře připojen text, jde o relaci subsumpce is-a. Na obr. 1.14, který znázorňuje ontologii před analýzou pomocí metodiky OntoClean, jsou tedy všechny relace typu is-a.



Obr. 1.14 Příklad neupravené ontologie.

1.12.4.1 Krok 1. Posouzení metavlastností jednotlivých tříd

Rigidita. Třídy Entita, Ocel, Brožura, Hřídel, Ocelový hřídel a Člověk jsou rigidní. Instance každé z těchto tříd musí zůstat instancí příslušné třídy za všech okolností. Proto tyto třídy označíme jako $+R$.

Třídu Řidič můžeme chápat dvojím způsobem, buď jako osobu, která právě řídí vozidlo, nebo jako osobu oprávněnou vozidlo řídit. V obou případech může instance této třídy přestat být řidičem. Také instance třídy Majitel vozidla mají dočasnou platnost.

Pokud majitel vozidlo prodá, daruje, nebo ho nechá sešrotovat, přestává být jeho majitelem. Proto jsou třídy Řidič a Majitel vozidla antirigidní a označíme je symbolem ~*R*.

Identita. Pro třídu Entita neexistuje jednotné kritérium identity, entitou může být cokoliv. Proto ji označíme –*I* a –*O*. Pro instance třídy Ocel definujeme kritérium identity pomocí hutnických norem oceli ČSN řady 41 (např. konstrukční ocel 11500). Třída Ocel tedy poskytuje kritérium identity, a proto ji označíme +*O* a +*I*.

Instance třídy Hřídel jsou identifikovány svými geometrickými rozměry. Ty se nedědí od hierarchicky nadřazené třídy Entita, a proto Hřídel označíme také +*O* a +*I*.

Třída Ocelový hřídel dědí kritéria identity od obou svých nadtříd. Objekty této třídy jsou identifikovány svými geometrickými rozměry a typem oceli, ze které jsou vyrobeny. Vlastní kritérium identity tato třída neposkytuje, pouze kritéria identity přenáší. Proto ji označíme –*O* a +*I*.

Třída Brožura je definována podle výkladového slovníku české terminologie z oblasti informační vědy a knihovnictví jako „Dokument o rozsahu nejméně 5 a nejvíce 48 stran, nepočítaje obálku". Zahrnuje tedy jakékoliv brožované knížky bez ohledu na jejich obsah. Jako kritérium identity můžeme tedy vzít geometrické rozměry brožury a počet stran. Toto kritérium třída zjevně nedědí od nadřazené třídy Entita, ale poskytuje je, a proto ji označíme +*O* a +*I*.

Ve třídě Návod k použití se definuje identita vymezením předmětu, který návod popisuje (co má uživatel používat) a cílové skupiny lidí, pro které je návod určen. Tato kritéria identity třída nedědí od třídy Brožura, a proto má metavlastnosti +*O* a +*I*.

Také třídy Řidič a Majitel vozidla nedědí kritérium identity. Pokud by pro objekty těchto tříd existovalo kritérium identity, musely by je tyto třídy poskytovat. Jak jsme však již ukázali, tyto třídy jsou antirigidní ~*R*. Proto nemohou poskytovat kritérium identity (viz definice pojmu Identita). Proto označíme tyto třídy ~*O* a ~*I*.

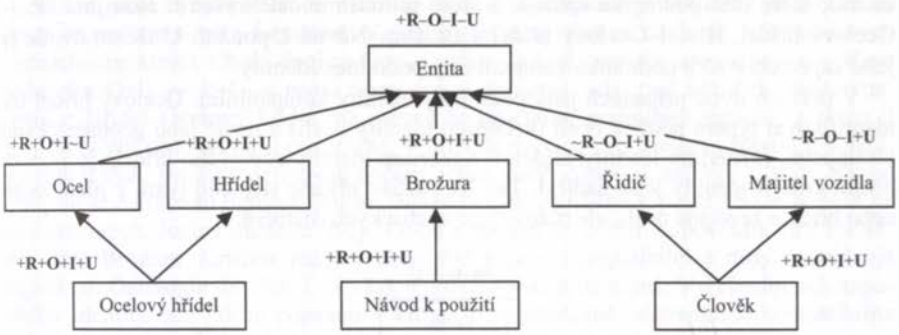
Ve třídě Člověk můžeme definovat kritérium identity například rodným číslem nebo profilem DNA. Toto kritérium není zděděné, a proto má tato třída metavlastnosti +*O* a +*I*.

Jednota. Třída Entita obsahuje mnoho různorodých objektů, a proto nelze určit podmínku jednoty platnou pro všechny. Proto řekneme, že tato třída nepřenáší jednotu. Protože však některé instance této třídy mohou být celkem, nepřiřadíme této třídě silnější metavlastnost antijednoty (~*U*), ale pouze –*U*.

Pro třídu Ocel neexistuje relace, která by definovala její instance jako celky. To je častá vlastnost tříd, které jsou popsány nepočitatelnými výrazy. Tyto výrazy jsou nepočitatelné právě proto, že objekty netvoří celky, které by bylo možné spočítat. Zde si je třeba uvědomit rozdíl mezi třídou Ocel (není celek) a třeba Ocelový plech (je celek). Podobně například třídy Voda nebo Sníh nedefinují celek, na rozdíl od tříd Vodní kapka nebo Sněhová koule. Třída Ocel proto přenáší antijednotu a tuto metavlastnost označíme ~*U*.

Ostatní třídy v uvedeném příkladu, tj. Ocelový hřídel, Hřídel, Brožura, Návod k použití, Řidič, Majitel vozidla a Člověk, tvoří celky, proto přenášejí jednotu a označíme je +*U*.

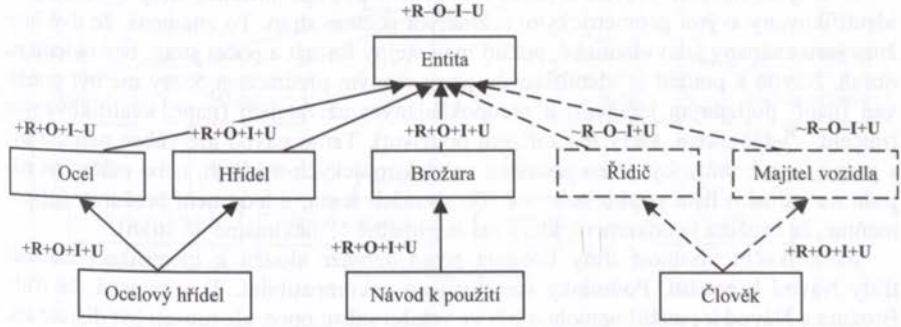
Výsledné přiřazení metavlastností třídám je znázorněno na obr. 1.15.



Obr. 1.15 Přiřazení metavlastností.

1.12.4.2 Krok 2. Rigidní třídy – vytvoření páteřní taxonomie

V tomto kroku se zaměříme na třídy definované pomocí rigidních vlastností, tj. rigidní třídy označené +R. Tyto třídy představují invariantní část ontologie. Taxonomii tvořenou pouze rigidními třídami nazveme páteřní taxonomie. Vzhledem k tomu, že každá třída, která poskytuje identitu (+O), musí být rigidní, zaručuje páteřní taxonomie, že budou zahrnuty všechny třídy, jež definují vlastnosti sloužící k rozlišení objektů. Páteřní taxonomie odpovídající řešenému příkladu je znázorněna na obr. 1.16 plnými čarami. Tento krok uplatňuje omezení č. 1 metodiky OntoClean.



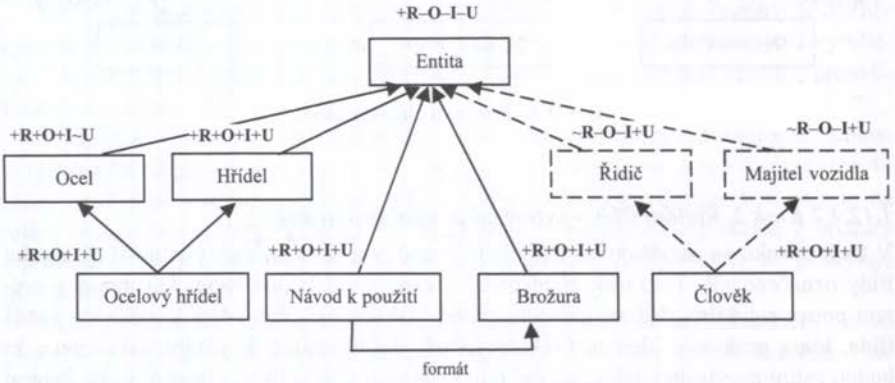
Obr. 1.16 Páteřní taxonomie.

1.12.4.3 Krok 3. Ověření kritéria identity

Jak jsme uvedli při analýze metavlastností, musí platit, že pokud je třída Q nadtřídou třídy P a třída Q přenáší identitu, musí ji přenášet i její podtřída P (platí tedy $Q{+}I \rightarrow P{+}I$). Navíc třídy, které mají nekompatibilní podmínky identity, musejí být disjunktní, tj. nemohou být ve vztahu subsumpce. Tyto podmínky vyjadřují omezení 2 a 5. V ontologii dle obr. 1.11 žádná dvojice tříd podmínku $Q{+}I \rightarrow P{+}I$ neporušuje. Jsou zde ale tři dvoji-

ce tříd, které tuto podmínku splňují, a proto je musíme dále vyšetřit. Jsou jimi Ocel–Ocelový hřídel, Hřídel–Ocelový hřídel a Brožura–Návod k použití. U těchto dvojic je ještě zapotřebí ověřit podmínku kompatibility podmínek identity.

V prvních dvou případech jsou podmínky identity kompatibilní. Ocelový hřídel lze identifikovat typem použité oceli (kritérium identity oceli) a navíc jeho geometrickými vlastnostmi (kritérium identity hřídele). Kritérium identity ocelového hřídele je sjednocením kritérií identity jeho nadtříd. Tak lze rozlišit hřídele stejného tvaru z různé oceli nebo hřídele ze stejné oceli, ale různých geometrických rozměrů.



Obr. 1.17 Upravená ontologie po ověření kritéria identity.

U dvojice Brožura–Návod k použití je situace odlišná. Instance třídy Brožura jsou identifikovány svými geometrickými rozměry a počtem stran. To znamená, že dvě brožury jsou chápány jako identické, pokud mají stejný formát a počet stran, bez ohledu na obsah. Návod k použití je identifikován popisovaným předmětem, který má být používán (např. pojízdným jeřábem) a předpokládaným uživatelem (např. kvalifikovaným řidičem – jeřábníkem, který má zařízení používat). Tento návod ale vůbec nemusí být v psané formě, může být třeba přenášen na elektronických médiích, nebo může být napsán na jediném listu papíru nebo na 100 stranách textu, a tedy není brožurou (připomeňme, že brožura je dokument, který má minimálně 5, maximálně 48 stran).

Identifikační vlastnost třídy Brožura proto nemůže sloužit k identifikaci instancí třídy Návod k použití. Podmínky identity jsou nekompatibilní. To znamená, že třídy Brožura a Návod k použití nemohou být ve vztahu subsumpce, ale musejí být disjunktní. Jedna z možností, jak toho dosáhnout, je ukázána na obr. 1.17. Třída Návod k použití je spojena s třídou Brožura relací 1:N s názvem „formát", která charakterizuje způsob publikace návodu.

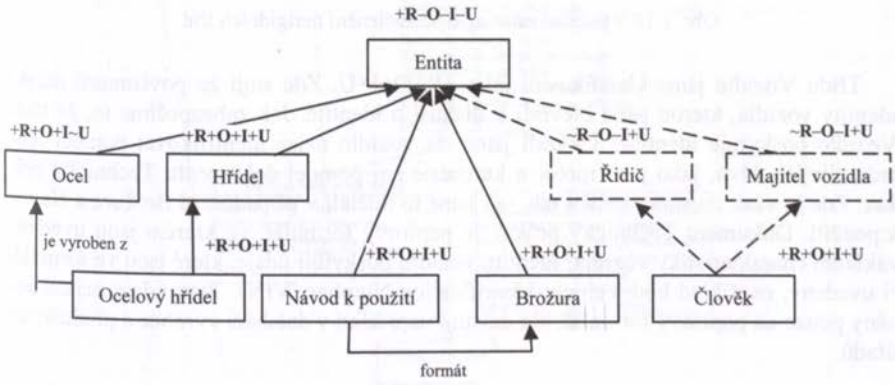
1.12.4.4 Krok 4. Ověření kritéria jednoty

Podobně jako v předchozím kroku musí platit, že pokud je třída Q nadtřídou třídy P (viz obr. 1.13) a třída Q přenáší jednotu, musí ji přenášet i její podtřída P (platí tedy Q + U → P + U). Navíc třídy, které mají nekompatibilní podmínky jednoty, musejí být disjunktní. Toto kritérium je vyjádřením omezení 3 a 5 metodiky OntoClean.

Návod může být rozdělen do několika dílů, které mohou být vytištěny jako brožury s různým počtem stran a formátem. V úvahu přicházejí dvě ze tří dvojic tříd popsaných v předchozím kroku. Chybí dvojice Ocel–Ocelový hřídel, protože tato dvojice nesplňuje podmínku Q+U → P+U, a proto nemá smysl zkoumat, zda mají být třídy disjunktní. Dvojice Hřídel–Ocelový hřídel má zjevně kompatibilní podmínky jednoty, a proto jí není zapotřebí věnovat další pozornost. V případě dvojice tříd Brožura–Návod k použití je celek u třídy Brožura definován počtem stran, popř. obálkou, zatímco u třídy Návod k použití je definován úplností popisu. Může se stát, že návod k použití bude popsán ve více brožurách. Jediná instance třídy Návod k použití by potom odpovídala více instancím třídy Brožura. Kritéria jednoty obou tříd jsou nekompatibilní a třídy musejí být disjunktní. Ontologie dle obr. 1.17 však tento nedostatek už nemá. Vyřešením nekompatibility identity tak, jak je popsáno v kroku 3, se současně odstranila nekompatibilita podmínek jednoty. Žádná další změna v tomto kroku proto není potřebná.

1.12.4.5 Krok 5. Ověření kritéria jednoty a antijednoty

V tomto kroku budeme testovat omezení 4, které říká, že jestliže třída Q přenáší antijednotu a Q je nadtřídou P, potom musí třída P přenášet antijednotu, Q ~ U → P ~ U. V ontologii dle obr. 1.18 je jedna dvojice tříd, která porušuje toto omezení. Jde o dvojici Ocel–Ocelový hřídel. Třída Ocelový hřídel přenáší jednotu, instance třídy Ocelový hřídel je celek. Každá instance třídy Ocelový hřídel je ale dle původního návrhu ontologie současně instancí třídy Ocel, a tedy není celkem. Toto je spor, který je vyjádřen právě omezením 4. Jedna z možností, jak upravit ontologii tak, aby toto omezení nebylo porušeno, je ukázán na obr. 1.18.



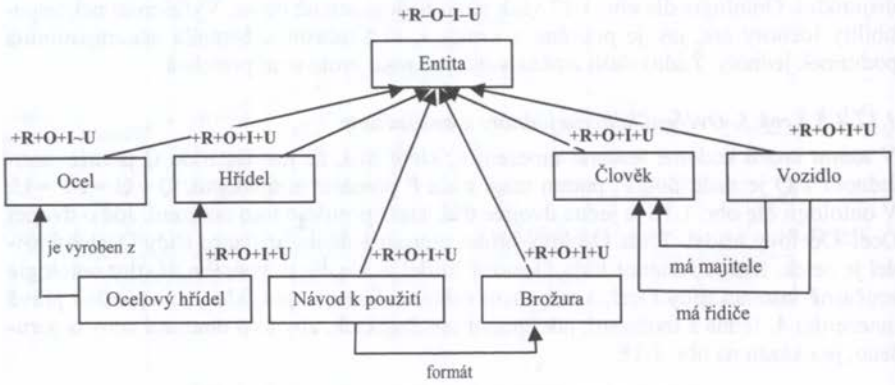
Obr. 1.18 Upravená ontologie po ověření kritéria jednoty a antijednoty.

Podstata navržené změny spočívá v nahrazení relace subsumpce mezi třídami jinou relací, v tomto případě relací „je vyroben z", na kterou se nevztahují uvedená omezení.

1.12.4.6 Krok 6. Začlenění nerigidních tříd

Ve druhém kroku jsme se zaměřili na rigidní třídy a pomocí nich jsme definovali páteřní taxonomii. V ontologii ovšem zbývají ještě třídy, které nejsou rigidní. Z omezení 1 víme, že rigidní nebo nerigidní třída nemůže být podtřídou nějaké antirigidní třídy.

V ontologii dle obr. 1.14 existují dvě dvojice tříd, které porušují toto omezení. Jsou to Řidič–Člověk a Majitel vozidla–Člověk. Třída Člověk je rigidní (+R) – instance této třídy nemůže přestat „být člověkem". Pokud by ale antirigidní třída Řidič byla nadtřídou třídy Člověk, musela by existovat možnost, jak by instance třídy Člověk přestala být instancí třídy Řidič při zachování uvedeného vztahu subsumpce. To je ale konflikt, kterým lze na tomto konkretním příkladu demonstrovat omezení 1. Tento konflikt odstraníme například zavedením další třídy Vozidlo, které bude mít se třídou Člověk vztah zprostředkovaný relacemi „má majitele" a „má řidiče". Výsledná ontologie je uvedena na obr. 1.19.



Obr. 1.19 Výsledná ontologie po začlenění nerigidních tříd.

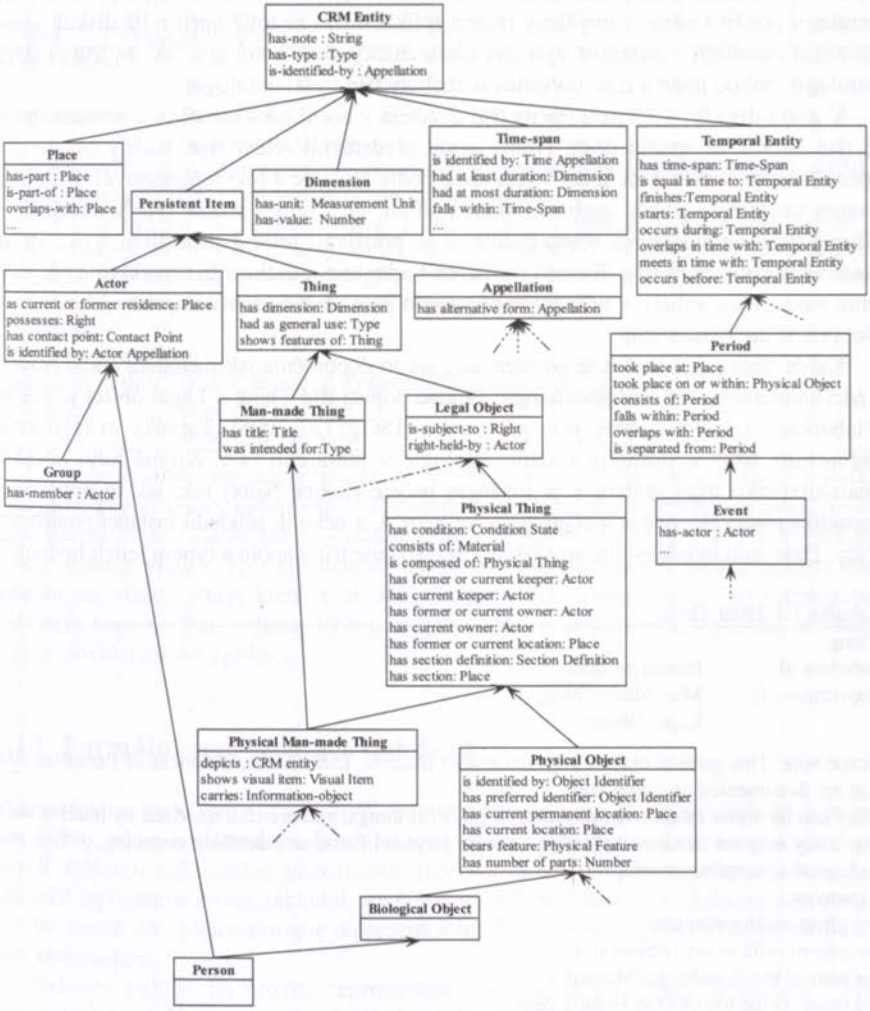
Třídu Vozidlo jsme klasifikovali jako +R+O+I+U. Zde stojí za povšimnutí otázka identity vozidla, kterou jsme otevřeli v diskusi o identitě. Jak zabezpečíme to, že třída Vozidlo poskytuje identitu? Ukázali jsme, že vozidlo nelze identifikovat pomocí jeho jednotlivých částí, jako jsou motor a karosérie ani pomocí dokumentu Technický průkaz. Zde je však musíme rozlišit tak, jak jsme to udělali v případě tříd Brožura a Návod k použití. Dokument Technický průkaz je papírový formulář, ve kterém jsou uvedeny základní charakteristiky vozidla. Identitu vozidlu poskytují údaje, které jsou ve formuláři uvedeny, například kód Vehicle Identification Number (VIN). Tyto údaje nejsou vázány pouze na papírový formulář, ale existují například v databázi výrobce a příslušných úřadů.

1.13 Příklad: Ontologie CIDOC CRM

Příklad části ontologie vytvořené na základě konceptuálního referenčního modelu CIDOC CRM je zobrazen grafem na obr. 1.20.

Tato ontologie obsahuje 81 tříd a 132 vlastností. Norma ISO 21127-2006 uvádí model CIDOC CRM jako standard k popisu kulturních statků, tedy pro aplikace v muzeích a galeriích. Při návrhu ontologie se autoři snažili co nejlépe zachovat požadované metavlastnosti relací definujících jednotlivé třídy. Třídy a jejich atributy jsou navíc na-

vrženy s ohledem na použití v předpokládaných aplikacích. V textu se používá anglická verze standardu. Existuje sice neoficiální český překlad, viz http://www.citem.cz/tag/cidoc-crm/, avšak ten vznikl podle autorů překladu pouze pro interní potřebu.



Obr. 1.20 Grafické znázornění části ontologie CIDOC CRM.

Na této ontologii lze ukázat některé dříve vysvětlené kroky a pojmy. Ontologie je tvořená hierarchií tříd is-a a současně binárními relacemi popisujícími vztahy mezi třídami. Podle míry formalizace (viz oddíly 1.2 a 1.14) jde o semiformální ontologii. Je na každém uživateli, aby v případě potřeby reprezentoval třídy a jejich vlastnosti formálními axiomy.

CIDOC CRM je sice doménová ontologie zaměřená na oblast kulturního dědictví, ale zahrnuje v sobě i třídy, které nacházíme v základních ontologiích. Není tedy zapotřebí rozšiřovat tyto ontologii mapováním na nějakou základní ontologii. Volba tříd a atributů většinou není příliš zatížená orientací na popisovanou oblast, a proto lze tuto ontologii použít i mimo zamýšlený rámec aplikací. Lze se totiž opřít o již diskutovanou vlastnost ontologií – možnost vytvářet hierarchicky nižší ontologie tak, že třídy nižších ontologií mohou dědit a dále upřesňovat atributy tříd vyšší ontologie.

V grafu dle obr. 1.20 jsou jména tříd uvedena v horní části rámečku a seznam atributů tříd (slotů) ve spodní části. Hrany grafu představují relaci is-a. Každý slot je dále specifikován (viz definice facet v metodice podle Noyové a McGuinnessové). Ontologie ovšem nevyužívá všech možností popisu slotu. Je uveden pouze typ hodnoty slotu. Kromě typů definovaných touto ontologií se používají ještě 3 primitivní typy: String, Number a Time Primitive. Rozsah možných hodnot není definován obvyklým způsobem (min value, max value), v některých případech jsou však hodnoty popsány jako zvláštní slot (viz třídu Time-span).

Každá třída a její atribut je popsána tak, jak to doporučuje jak metodika podle Noyové a McGuinnessové, tak i Methontology. Příklad popisu tříd Thing a Legal object je uveden v tabulkách 1.1 a 1.2, které jsou převzaty z (ISO21127-2006). Tabulky uvádějí vztah definované třídy k ostatním třídám ontologie v hierarchii is-a. Kromě toho obsahují charakteristiku třídy textem v přirozeném jazyce (Scope Note) tak, jak to doporučuje metodika podle Noyové a McGuinnessové bodě 4, a několik příkladů instancí popisované třídy. Dále jsou uvedeny atributy (sloty) popisované třídy spolu s typem jejich hodnot.

Tabulka 1.1 Třída Thing

Thing	
Subclass of:	Persistent Item
Superclass of:	Man-Made Thing
	Legal Object
Scope note: This general class comprises usable discrete, identifiable, instances of Persistent Item that are documented as single units. They can be either intellectual products or physical things, and are characterized by relative stability. They may for instance either have a solid physical form, an electronic encoding, or they may be logical concept or structure. Examples: my photograph collection the pint of milk in my refrigerator the plan of the Stassburger Muenster the thing on the top of Otto Hahn's desk the design of the no-smoking sign the cave of Dirou, Mani, Greece	
Properties	has dimension: Dimension had as general use: Type shows features of: Thing

Ontologii CIDOC CRM popisujeme podrobněji, včetně tabulek 1.1 a 1.2, také proto, aby mohl čtenář sám posoudit, zda taxonomie této často užívané ontologie splňuje kritéria OntoClean, která jsme uvedli v oddílu 1.12 a jejich použití předvedli na jednoduché ontologii dle obrázků 1.14 až 1.19.

Tabulka 1.2 Třída Legal Object

Legal Object	
	Thing
Superclass of:	Physical Thing
	Information Object
Scope note: This class comprises those material or immaterial items to which instances of E30 Right, such as the right of ownership or use, can be applied.	
This is true for all Physical Thing. In the case of instances of Conceptual Object, however, the identity of the Conceptual Object or the method of its use may be too ambiguous to reliably establish instances of Right, as in the case of taxa and inspirations. Ownership of corporations is currently regarded as out of scope of the CRM.	
Examples:	
the Cullinan diamond	
definition of the CIDOC Conceptual Reference Model Version 2.1	
Properties:	is subject to: Right
	right held by: Actor

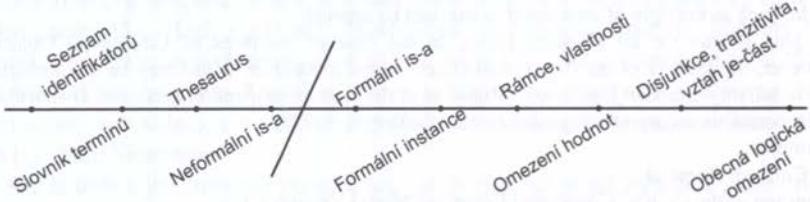
Přestože byla tato ontologie navržena s cílem standardizovat popis kulturních statků, jsou její nejvyšší vrstvy definovány velmi obecně a ontologii lze použít i mimo původně zamýšlenou oblast. Zaměření na kulturní oblast je zřejmé zejména z pojmenování tříd a slotů nižších vrstev. To však nemusí být na závadu. Jednak lze využít obecné části a definovat vlastní třídy, které z ní dědí, jednak některé prostředky pro reprezentaci ontologií, např. OCML (Motta, 1999), umožňují přejmenovat jména slotů tak, aby sloty lépe odpovídaly nové aplikaci.

1.14 Formální reprezentace ontologií

V předchozím textu jsme uvedli čtyři úrovně reprezentace ontologií (Uschold, Gruninger, 1996), od neformálních po formální reprezentace. Až doposud jsme ontologie popisovali většinou neformálně přirozeným jazykem nebo nejvýše semiformálně pomocí grafické reprezentace (viz základní ontologie a CIDOC CRM). Pro strojové zpracování je však nutné, aby byla ontologie reprezentována formálním jazykem s přesně definovanou sémantikou.

Podobný pohled na úrovně reprezentace jako práce (Uschold, Gruninger, 1996) poskytuje obr. 1.21. Na straně nejnižší formálnosti stojí popis slovníku, tedy popis významu použitých symbolů, a to prostřednictvím přirozeného jazyka nebo dokonce jen samotný seznam symbolů, kde význam lze odhadovat pouze z významu slova v přirozeném jazyce. Takto popsané slovníky občas obsahují snahu o vyjádření vztahů mezi symboly, jako například vztah is-a, nicméně stále platí, že popis je pouze v jazyce, který není formálně definován. Tyto úrovně jsou přinejmenším v současné době pro počítač nesrozumitelné a takovýto popis se za ontologii nepovažuje.

Bodem, od kterého se začíná mluvit o ontologii, tedy o skutečně formálním popisu konceptualizace, jsou jazyky, které formálně a s definovanou sémantikou vyjadřují alespoň jeden vztah mezi symboly. Typickými jsou vztahy: je-podtřída, je-instance apod. Popis, který dává do vztahu třídy a podtřídy ve smyslu vyjádření podmnožiny instancí, se nazývá taxonomie. Jako příklad taxonomie si uveďme botanickou taxonomii. Taxonomie často tvoří takzvanou páteř ontologie, tj. ontologii tvoří hierarchie tříd a vlastností a ty jsou doplněny omezeními hodnot vlastností, bližšími specifikacemi vlastností atd. Typickým představitelem prostředku pro popis ontologií na této úrovni jsou rámce, které umožňují formálně popsat třídy, jejich vlastnosti a jejich instance.



Obr. 1.21 Různé úrovně formálnosti a expresivity popisu ontologií.

O úroveň výše stojí další modely, které k taxonomii umožňují přidat další omezení, například omezení možných hodnot nebo bližší specifikace vlastností. Jako příklady takovýchto omezení si uveďme třeba omezení počtu rodičů jedince, vyjádření funkčnosti relace (např. má-rodné-číslo je funkční vlastnost, neboť člověk nemůže mít dvě rodná čísla), vyjádření inverznosti dvou relací (např. pro relace je-potomek a je-rodič) atd. Jako představitele této úrovně si uvedeme deskripční logiky, což jsou logiky poskytující prostředky potřebné pro praktické modelování znalostí, ale které jsou omezeny ve vyjadřovacích prostředcích tak, aby výsledné ontologie byly použitelné v praxi pro automatické uvažování. Tím se myslí, že k bezespornosti a úplnosti uvažování požadujeme též rozhodnutelnost, pokud možno řešitelnou v přijatelném čase.

Na konci pomyslné osy formálnosti popisu stojí ničím neomezené logiky prvního a vyšších řádů, modální logiky a další. Tyto jazyky poskytují velmi mocné modelovací prostředky, ve kterých lze vyjádřit skoro cokoliv, tedy popsat jakoukoliv logickou teorii. Nevýhodou je ale často nemožnost použít celý popis strojově, protože uvažování je v nich nerozhodnutelné.

Přes tuto nevýhodu zní obecné doporučení pro modelování ontologií, abychom používali jazyk, který nás v ničem neomezuje a který nám umožňuje vyjádřit vše, co potřebujeme. Pro praktické použití na různých úrovních pak použijeme jen část popisu, který jsme schopni zpracovat. Příkladem jsou ontologie, jako je Cyc a SUMO, které jsou popsány ve velmi expresivních jazycích CycL a KIF, nicméně existuje jejich export do jednodušších jazyků, jako je například OWL (Web Ontology Language, viz další výklad). Přístup použití jen části popisu lze ilustrovat i na konkrétním popisu ontologie v OWL, kde různí agenti mohou tento popis zpracovávat s použitím jen omezené množiny konstruktů v daném popisu – více o tom najdeme v oddílu o sémantickém webu.

Osa na obrázku nám tedy ukazuje uspořádání dle úrovně formalizace. Čím vyšší úroveň formalizace, tím spíše lze ontologii použít pro sdílení či automatické uvažování, neboť tím spíše je možné ontologii strojově zpracovávat. Na druhou stranu, tím těžší je také ontologii vyvinout. Dnes se nejčastěji jako ontologie pro omezenou doménu nebo pro konkrétní aplikaci používá nějaké vyjádření taxonomie obohacené o další omezení, jako například omezení hodnot či relací.

Existuje velký počet formálních jazyků pro reprezentaci znalostí a většinu z nich lze použít také pro reprezentaci ontologií. Uvedeme stručně několik nejvýznamnějších.

1.14.1 Rámcové modely

Systémy založené na rámcích používají jako modelovací primitiva rámce a jejich vlastnosti. Rámce byly představeny již v předchozích dílech této série knih (Mařík a kol., 1993), proto zde uveďme jen velmi stručnou charakteristiku. Ústředním modelovacím primitivem je rámec. Rámec může reprezentovat třídy nebo objekty. Rámec typicky obsahuje sloty – definice vlastností rámce včetně možnosti jejich omezení. Sloty jsou definovány pro konkrétní rámce a mohou nabývat hodnot jako například primitivní objekty (čísla nebo řetězce), jiné rámce nebo seznam rámců. Omezení, která jsou součástí rámců, mohou mít například formu šablon (definice výchozích hodnot).

Charakteristickou vlastností rámcových systémů je možnost definovat dědičnosti mezi rámci – rámce tak mohou dědit sloty včetně jejich omezení – a dále možnost definovat vztahy is-a mezi rámci, tedy vyjádřovat příslušnosti objektu ke třídě. Pro více detailů a ilustrační příklady odkazujeme čtenáře na knihu (Mařík a kol., 1993).

1.14.2 Sémantické sítě

Sémantická síť je graf, kde uzly jsou koncepty a kde hrany jsou relace mezi uzly. Sémantická síť může být použita jak pro vyjádření informace bez zvláštní ontologické struktury, tak pro vyjádření ontologie. Sémantická síť na úrovni ontologie vyjadřuje slovník, který je čitelný a užitečný především pro člověka. Obecné sémantické vztahy používané v takovýchto slovnících byly již uvedeny v předchozím textu, například v oddílu 1.4. Nejvíce používané relace mezi koncepty A a B lze shrnout takto:

- Synonymum – koncept A vyjadřuje to samé jako koncept B (např. hezký je synonymum k pěkný).
- Antonymum – koncept A vyjadřuje opak konceptu B (např. velký je antonymum k malý).
- Meronymum – koncept A je částí konceptu B (např. k domu je meronymem okno).
- Holonymum – koncept A má jako svou část koncept B (part-of, např. k domu je holonymum město).
- Hyponymum (nebo troponymum) – A je druhem B (is-a, např. pes je hyponymem zvířete).
- Hyperonymum – A je nadtypem B (např. zvíře je hyperonymem psa).

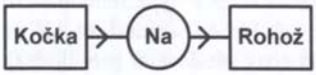
Mimo tyto základní vztahy se používají i speciální vztahy závislé na doméně, třeba žije-v (např. delfín žije-v voda). Tyto a další vztahy jsou často zamýšlené pouze pro použití člověkem, jejich formální reprezentace je problematická, popřípadě by jejich formální vyjádření mohlo být ve sporu se zamýšlenou sémantikou již existující sítě.

Proto je sporné, zda konceptualizaci vyjádřenou sémantickou sítí bez formálně definované sémantiky můžeme ještě považovat za ontologii. Poznamenejme k tomu, že sice nelze provádět plné uvažování v rámci formální logiky, nicméně lze často například ověřit relaci subsumpce mezi dvěma koncepty (pomocí tranzitivity uvedených relací). Přes tento nedostatek mají sémantické sítě svou oblast použitelnosti, například v hypertextových systémech.

Sémantické sítě se objevily jako první pokusy o vyjádření mezijazyka (interlingua) pro překlad mezi lidskými jazyky a jako nástroj se pro tento účel používají i dnes. Typickým představitelem konkrétní takovéto „ontologie" je WordNet (Miller, 1998), který popisuje vztahy mezi anglickými slovy a uvádí definici pojmů prostřednictvím přirozeného jazyka – viz též předchozí výklad v této kapitole.

1.14.3 Konceptuální grafy

Konceptuální grafy (Conceptual Graphs, CG), (Sowa, 2000), je logický formalismus, který zahrnuje třídy, relace, individua a kvantifikátory. Tento formalismus vychází ze sémantických sítí, ale existuje pro něj přímý překlad do jazyka predikátové logiky prvního řádu, odkud přebírá i svou sémantiku. Hlavním rysem je standardizovaná grafická podoba reprezentace, která podobně jako v případě sémantických sítí umožňuje člověku rychlý náhled na to, co graf vyjadřuje. Konceptuální graf je bipartitní orientovaný graf, kde instance konceptů jsou zobrazeny obdélníkem a konceptuální relace pak oválem nebo kružnicí, hrany tyto uzly propojují s udáním směru relace. Relace může mít více než dvě hrany, v tom případě se hrany číslují. Příklad grafické reprezentace, čili takzvané Display Form (DF), věty „(nějaká) kočka je na (nějaké) rohoži" je zobrazen na obr. 1.22.



Obr. 1.22 Jednoduchý konceptuální graf v grafické reprezentaci DF.

Pomocí textové notace Linear Form (LF) se tato věta zapíše jako [Kočka]-(Na)-[Rohož]. DF a LF slouží pro prezentaci člověkovi; existuje též formální jazyk CG Interchange Form (CGIF), kde tato věta je zapsána jako [Kočka: *x] [Rohož: *y] (Na ?x ?y), kde *x je definice proměnné a ?x je odkaz na definovanou proměnnou, popř. za použití syntaktických zkratek – (On [Cat] [Mat]). Převod mezi těmito formáty je definován a je též definován převod z CGIF do jazyka KIF (viz další výklad) – uvedený příklad by byl vyjádřen jako (exists ((?x Cat) (?y Mat)) (On ?x ?y)). Přestože všechny formáty vypadají jinak, všechna vyjádření mají stejnou sémantiku v predikátové logice, a to (∃x:Cat) (∃y:Mat)on(x,y). Dle publikace (Sowa, 2000) mají konceptuální grafy stejnou vyjadřovací schopnost jako predikátová logika.

Za zmínku stojí další možnosti konceptuálních grafů – možnost definice konceptu pomocí jiného „vnořeného" konceptuálního grafu nebo možnost jednoduchého popisu *n*-árních (nejen binárních) relací.

1.14.4 Knowledge Interchange Format (KIF)

Jazyk KIF (Genesereth, Fikes, 1992) byl navržen v době počátku devadesátých let v rámci iniciativy KSE. Cílem bylo usnadnit vzájemné porovnávání, využívání a sdílení znalostí vyvinutých v různých znalostních systémech. KIF má deklarativní sémantiku, k vyhodnocení výrazů v jazyce KIF není zapotřebí interpreter, který by výrazy interpretoval. V původní verzi neexistoval softwarový nástroj, který by umožnil operační využití znalostí zapsaných v jazyce KIF – nebylo to ani cílem návrhu tohoto jazyka.

KIF vychází z jazyka predikátové logiky 1. řádu. Základními výrazy jsou sentence (uzavřené formule jazyka predikátové logiky), definice a pravidla (rule). Definice slouží k pojmenování objektů, funkcí a relací. Pravidla mohou být zapsána ve tvaru používaném pro dopředné řetězení (např. OPS 5) nebo pro zpětné řetězení tak, jak ho používá například Prolog. Výrazy, které označují proměnné, mají jako první znak otazník. Výrazy jazyka KIF používají prefixovou syntax, odvozenou z jazyka Lisp, ve tvaru (funktor argument-1 argument-2 ...). Výraz pro sečtení čísel 2 a 3 bude tedy (+ 2 3).

Nad rámec jazyka predikátové logiky 1. řádu obsahuje KIF některá rozšíření. Ta zahrnují možnost zpracovávat metaznalosti, tj. chápat znalosti zapsané v jazyce KIF jako samostatné objekty jazyka. Dále podporuje používání nemonotónních inferencí. KIF sám neobsahuje žádný inferenční mechanismus, poskytuje však vyjadřovací prostředky pro reprezentaci znalostí potřebných k odvozování.

Jak již z názvu jazyka vyplývá, zamýšlené použití bylo pro výměnu znalostí, nikoliv pro efektivní reprezentaci ontologií. Pro výměnu znalostí pomocí jazyka KIF mezi různými formalismy se volí následující postup. Nejprve se v jazyce KIF implementuje daný formalismus (např. rámce) a tato implementace se pak použije pro reprezentaci konkrétních znalostí nebo ontologie. To je také přístup, který byl využit v projektu Ontolingua. Reprezentace v jazyce KIF se pak může použít pro převod do jiné podoby reprezentace.

1.14.5 Ontolingua

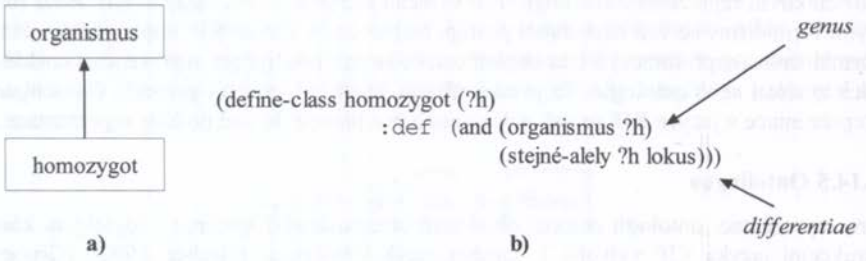
Pro reprezentaci ontologií pomocí objektově orientovaného systému s logickými konstrukcemi jazyka KIF vytvořil T. Gruber jazyk Ontolingua (Gruber, 1992), (Gruber, 1993). Prostředky jazyka Ontolingua lze použít k formální reprezentaci ontologií, také nám umožňují překládat ontologie do několika dalších aplikačních jazyků. Následující text si neklade za cíl poskytnout čtenáři úplný popis jazyka Ontolinga – chceme pouze ukázat některé jeho typické konstrukce.

Základní objekty jazyka Ontolingua jsou třída, funkce, relace, objekt a teorie (class, function, relation, object a theory). Jako příklad uveďme definici třídy:

```
(define-class <jméno-třídy> (<proměnná>)
        (<dokumentační řetězec>)
        :defnebo :iff-def <sentence v jazyce KIF s proměnnými>
        :constraints <sentence v jazyce KIF s proměnnými>
        :sufficient <sentence v jazyce KIF s proměnnými>
        :equivalent <sentence v jazyce KIF s proměnnými>
        :class-slots <specifikace slotu>
        :instance-slots <specifikace slotu>
        :axiom-def <sentence v jazyce KIF bez proměnných>)
```

V této definici výraz <jméno-třídy> pojmenovává definovanou třídu. Výraz <proměnná> má například tvar ?instance a označuje proměnnou, jejíž hodnotou je instance definované třídy. Tak mohou sentence v definici třídy odkazovat k jejím instancím. Výraz <dokumentační řetězec> je komentářem v přirozeném jazyce, kterým lze verbálně popsat definici třídy. Tento text je dobrovolný a dále se nezpracovává. Následují sentence v jazyce KIF, jejichž role je popsána klíčovým slovem, které začíná dvojtečkou. Konkrétní definice většinou neobsahuje všechna možná klíčová slova a sentence, v definici třídy jsou použity ty konstrukce, které jsou pro definování třídy zapotřebí.

Sentence označená klíčovým slovem :def vyjadřuje nutnou definiční podmínku, kterou musejí instance třídy splnit. Pokud je tato podmínka nutná a postačující, je označena jako :iff-def. Klíčová slova :constraints a :equivalent označují také nutnou, popř. nutnou a postačující, podmínku, ale v tomto případě nejde o podmínky, které bychom při konceptualizaci označili jako definiční. Klíčové slovo :sufficient označuje postačující podmínku, kterou musejí instance splňovat. Konečně jako :axiom-def jsou uvedeny sentence, které se vztahují na celou třídu a nejenom na její jednotlivé instance. Proto v těchto axiomech nevystupuje proměnná ?instance. Příkladem může být axiom, definující subsumpci tříd. V definici třídy Homozygot by mohl být axiom ve tvaru :axiom-def (subclass-of Homozygot Organismus). Subsumpce ale také může vyplývat přímo z definiční sentence, která popisuje třídu. Ontolingua vytvoří relaci subclass-of, pokud najde unární relaci v sentenci označené :def nebo :iff-def. Unární relaci chápe jako genus a binární relace tvoří differentiae, viz obr. 1.23.



Obr. 1.23 (a) Hierarchie is-a dvou tříd, (b) definice v jazyce Ontolingua.

K popisu tříd a jejich instancí lze použít sloty, které reprezentují binární relace, do nichž mohou vstoupit jak třídy, tak i jejich instance. Příkladem binární relace, do které vstupují třídy, je uvedená relace subclass-of. Kromě uvedené definice pomocí klíčového slova :axiom-def a implicitní konstrukce dle obr. 1.23 nabízí Ontolingua možnost definovat sloty tříd pomocí klíčového slova :class-slots. Lze tedy psát :class-slots (subclass-of Homozygot Organismus). Jde však pouze o zvyklost z jazyka rámců, Ontolingua reprezentuje vnitřně specifikaci slotů třídy stejně jako sentence označené klíčovým slovem :axiom-def.

Podobně lze vyjádřit sloty instancí. Pokud jsou tyto sloty součástí definice, lze je uvést jako součást sentence s klíčovým slovem :def nebo :iff-def (viz slot stejné-alely u třídy homozygot, obr. 1.23). V případě, že nejde o definiční sloty, jako je tomu u následující třídy vozidlo (třída vozidlo není definována tím, že její instance mají řidiče a majitele), měly by být tyto sloty uvedené s klíčovým slovem :constraints.

```
(define-class vozidlo (?voz)
        :constraints (and (má_majitele ?voz Člověk)
                          (má_řidiče ?voz Člověk)))
```

Analogické je to se sloty tříd, neboť Ontolingua nabízí možnost definovat tyto sloty pomocí klíčového slova :instance-slots. Vnitřně jsou však reprezentovány jako sentence označené klíčovým slovem :constraints.

Ontolingua nabízí bohatý soubor vyjadřovacích prostředků, kterými lze formálně reprezentovat ontologie. Na http://www-ksl.stanford.edu/htw/dme/thermal-kb-tour/ONTOLINGUA.html lze najít podrobný manuál jazyka Ontolingua, několik příkladů je uvedeno v práci (Goméz-Pérez a kol., 2004). Motta (1999) popisuje jazyk OCML (Operational Conceptual Modelling Language), který implementuje většinu prostředků jazyka Ontolingua a navíc je vybaven odvozovacím algoritmem pro přímé řetězení (OPS 5), zpětné řetězení (Prolog) a dědění v hierarchii is-a.

1.14.6 Deskripční logiky

Deskripční logiky (Description Logics, DL) jsou logiky, jejichž konstrukty slouží primárně pro formální popis konceptů a relací (Baader a kol., 2003). Tyto logiky vycházejí ze snah formalizace a řešení dalších problémů při použití sémantických sítí a systémů založených na rámcích. Často sémanticky vycházejí z predikátové logiky prvního řádu, ale jejich slovník je volen tak, aby stačil pro v praxi používané popisy, a zároveň, aby výsledná logika měla dobré výpočetní vlastnosti (typickým požadavkem je alespoň rozhodnutelnost v konečném čase). Výzkum v této oblasti se mimo jiné zabývá tím, jak jsou použitelné jednotlivé konstrukty logik pro reálné aplikace a jaký dopad mají tyto konstrukty na možnost použití popisů v uvažování při „dobré" komplexnosti algoritmu rozhodnutelnosti.

Popis ontologie v deskripční logice tedy používá konstrukty, které mají jasnou sémantiku danou v predikátové logice. Kvůli historickému vývoji se však používá odlišná notace, která je blíže sémantickým sítím a rámcovým systémům. Uveďme si jako příklad zápis v logice AL (attributive language), což je minimální deskripční logika s v praxi použitelným slovníkem. Předpokládejme, že *Osoba* a *Žena* jsou atomické koncepty. Pak Muž může být definován jako $Muž \equiv Osoba \sqcap \neg Žena$. Dále předpokládejme, že *máDítě* je atomická role. Pak $Person \sqcap \exists máDítě.\top$ je množina všech osob, které mají dítě. Dále $Osoba \sqcap \forall máDítě.Žena$ jsou všechny osoby, jejichž děti jsou pouze ženy. V systému deskripční logiky se obecně odděluje část TBox (terminological box), která zahrnuje popis terminologie, zejména tedy vymezuje vztahy mezi koncepty a rolemi, a část ABox (assertional box), která zahrnuje popis stavu světa za pomocí zavedené terminologie, tedy vztahy mezi individui a koncepty a dále vztah mezi individui definovaný rolemi. TBox tedy v podstatě odpovídá ontologii a ABox (popř. sjednocení částí ABox a TBox) odpovídá znalostní bázi[10]. Pomocí terminologie uvedené v předchozím příkladu lze v části ABox například říci, že *jan:Muž*, tedy že konkrétní individuum *Jan* patří do třídy *Muž*.

[10] Toto rozdělení nelze brát dogmaticky, ontologie v některých interpretacích též může obsahovat individua – příkladem je geografická ontologie, která zahrnuje popis konkrétních států, viz též poslední krok v oddílu 1.10.1.

Protože motivem pro zavedení deskripčních logik je zejména možnost uvažování (reasoning), uveďme zde několik příkladů toho, co se od automatizovaného uvažování při zpracování ontologií typicky očekává. Poznamenejme rovnou, že tyto úlohy jsou (i v jiných formálních reprezentacích) převoditelné na nalezení důsledku teorie tvořené ontologií, popř. ontologií a znalostní bází. Prvním příkladem je ověření subsumpce, tedy toho, zda je nějaká třída podtřídou jiné třídy, nebo jinak řečeno, zařazení třídy do taxonomie. Často je jako jedním z hlavních nástrojů pro vytváření ontologií právě výpočet a zobrazení hierarchie tříd a vlastností. Dalším příkladem je ověření konzistence ontologie. Při návrhu větší ontologie se toto ověření používá pro ověření použitelnosti ontologie. Často je též požadována detekce problematických částí, které způsobují nekonzistentnost ontologie, aby autor ontologie mohl tyto části upravit. Dále chceme často ověřit, zda je nějaké tvrzení důsledkem ontologie, můžeme chtít nalézt všechny důsledky ontologie. Jako poslední typický příklad uveďme ověření příslušnosti individua (z ABoxu) do určené třídy (z TBoxu).

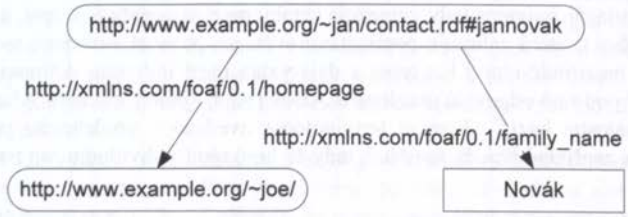
Deskripčním logikám je věnována pozornost v navazujících kapitolách této knihy, zejména v celé kapitole 3, ale i v dalších, protože jazyk OWL používaný v oblasti sémantického webu je na deskripční logice založen.

1.14.7 Jazyky sémantického webu (RDF, RDFS, OWL)

Významný impuls do vývoje ontologií přinesly aplikace v oblasti sémantického webu. Rozšíření primárních zdrojů na webu (tj. zejména webových stránek) o metadata, která je popisují, umožňuje lepší strojové zpracování informací na webu. Pro formální reprezentaci sémantiky těchto metadat existuje několik jazyků na různé úrovni, přičemž jednou ze základních myšlenek je definice významu metadat pomocí sdílených ontologií.

Základním jazykem pro popis internetových zdrojů je jazyk Resource Description Framework (RDF). Základní konstrukcí tohoto jazyka (Manola a kol., 2004) je trojice subjekt–predikát–objekt, kde každý element trojice může být webovým zdrojem (identifikovaným pomocí URL) a objekt dále může být literálem (řetězcem, číslem, datem, ...). Interpretací takovéto trojice může být též subjekt–vlastnost–hodnota vlastnosti. Příklad vyjádření informace „Jan-Novák má příjmení Novák a uvedenou domovskou stránku" ve dvou trojicích se stejným subjektem je ilustrován na obr. 1.24. Pro strojovou výměnu dokumentů v RDF se používá speciální zápis v jazyce XML.

V předchozím příkladu jsme viděli použití termínů vyjadřující vlastnosti „má příjmení" a „má domovskou stránku". Jazyk RDF umožňuje tyto termíny používat, ale sám



Obr. 1.24 Příklad vyjádření informací v jazyce RDF.

o sobě neumožňuje jejich vymezení. Pro popis termínů se na sémantickém webu používají jazyky RDFS a OWL.

Jazyk RDF Schema (RDFS) slouží pro popis taxonomie objektů a vlastností, dále umožňuje pro vlastnosti definovat definiční obor a obor hodnot (Brickley a kol., 2004). Tyto možnosti často pro jednodušší aplikace mohou stačit. Pro komplexnější popis ontologií slouží jazyk Web Ontology Language (OWL). Jazyk OWL (McGuinness a kol., 2004) je založen na deskripční logice, má tedy mocnější vyjadřovací schopnosti při zachování výpočetních vlastností pro uvažování, které mají zaručit jeho použitelnost pro praktické aplikace. Oproti RDFS umožňuje například specifikovat kardinalitu hodnot vlastností, říci, že dvě třídy jsou disjunktní, tranzitivitu rolí apod. Zápis ontologie v jazyce OWL může být proveden opět v jazyce RDF.

Lze tedy shrnout, že jazyk RDF je základním stavebním kamenem sémantického webu, který umožňuje vyjadřovat a slučovat popis v podobě binárních relací. Jazyky RDFS a OWL zavádějí do RDF konstrukce umožňující specifikovat ontologie, a tím umožňují výměnu informací včetně vymezení jejich významu. Podrobně se těmito jazyky a sémantickým webem obecně zabývá kapitola 2.

1.15 Příklady aplikací

Základní motivace pro použití ontologií již byla uvedena v předchozím textu včetně několika ukázkových aplikací. Je zřejmé, že aplikace jako formální kontrola konzistence návrhu ontologie, znovupoužití a integrace systémů včetně multiagentních systémů, anotování a následné vyhledávání atd. jsou společné pro mnoho oblastí. Obory, kde jsou ontologie využívány, zahrnují biomedicínu, obchod, různé oblasti průmyslu, knihovnicko-informační odvětví, zeměpis, právo, softwarové inženýrství atd. Podívejme se v této sekci jen v krátkosti na některé praktické aplikace ontologií ve vybraných oborech.

Oborem, který poskytuje plno impulsů pro vývoj ontologií a znalostních systémů obecně, je biomedicína. Jako příklad aplikace z této oblasti lze uvést test konzistence velké ontologie Medical Entities Dictionary (MED), která obsahuje 100 210 tříd a 261 vlastností. Tato ontologie byla zkonvertována do formální podoby v OWL (Kershenbaum a kol., 2006) a zkontrolována inferenčním strojem. Při tomto automatickém testu se zjistilo, že ontologie obsahuje systematické chyby v modelování a významné množství chybějících relací podtřída. Kdyby tyto problémy nebyly odstraněny, vedlo by to dle Kershenbauma (Kershenbaum a kol., 2006) k chybějícím výsledkům v systémech pro podporu rozhodování a v systémech pro kontrolu infekcí, které běžně ontologii MED používaly při prohlídkách pacientů.

Dalším příkladem je využití v projektu Eurogene (Zdrahal a kol., 2009). Eurogene je projekt podporovaný Komisí Evropských společenství (CEC), jehož cílem je podpora sdílení mnohojazyčných multimediálních výukových materiálů v oblasti genetiky člověka. Komunitu uživatelů tvoří jednak poskytovatelé, kteří zpřístupňují své přednášky a jiné výukové materiály, jednak uživatelé těchto materiálů. Poskytovateli jsou většinou vysokoškolští pedagogové a výzkumníci v oboru, uživateli jsou jednak pedagogové, kteří chtějí využít zkušeností svých kolegů, jednak studenti. Výukové materiály se strojově překládají do osmi jazyků, pokud existuje mezi jazykem zdrojového textu a cílo-

vým jazykem překladový software. Ten poskytuje francouzská firma SYSTRAN, která je jedním z partnerů projektu.

Sdílení je podporováno dvěma ontologiemi. Poskytovatelé nahrají svou přednášku na internetový portál projektu. Pokud je přednáška ve strojově čitelném formátu (Word, pdf, PowerPoint apod.) je automaticky anotována pomocí ontologie genetických pojmů. Ta byla vyvinuta společným úsilím skupiny genetiků a znalostních inženýrů. Je zaměřena na výuku genetiky a v době vzniku knihy obsahovala 2100 pojmů reprezentovaných více než 18 000 terminologickými výrazy v devíti evropských jazycích. Terminologie obsahuje také synonyma, pokud jsou v daném jazyce používány. Tato ontologie obsahuje jednak relace subsumpce (is_a), jednak blíže nespecifikované relace „Related Term". Pokud není přednáška ve strojově čitelném formátu (video, obrázky, audio), musí poskytovatel přednášku anotovat sám, má však podporu nástroje, který mu dovoluje volit pojmy z ontologie. Druhá ontologie definuje témata z oblasti genetiky, např. genetika je na nejvyšší úrovni rozdělena na statistickou, klinickou (lékařskou) a molekulární. Tato ontologie je popsána pouze výrazy v angličtině a obsahuje 171 hierarchicky uspořádaných témat. Protože tématy jsou abstraktnější pojmy, které nemusejí být explicitně vyjádřeny textem prezentace, anotuje nabídnutý výukový materiál vždy jeho autor.

Ontologie se v této aplikaci používají několika způsoby. Především umožňují vyhledávat dokumenty v mnohojazyčném prostředí. Protože jsou explicitně reprezentovany pojmy (významy) a jejich terminologickou reprezentací v různých jazycích, lze formulovat dotaz v jednom jazyce a požadovat materiál v jazyce jiném. Kromě toho mnohojazyčnou ontologii využívá software pro strojový překlad tím, že nahrazuje výrazy používané v běžném jazyce výrazy doménově závislými. Anotace pomocí mnohojazyčné ontologie také umožňuje vyhledávat sémanticky podobné materiály. Podobnost se vyjadřuje zobrazením anotací do vektorového prostoru a určením kosinu mezi vektory. V době vzniku textu obsahovala databáze portálu Eurogene přibližně 2500 výukových dokumentů v devíti jazycích.

Druhým oborem, který je důležitý nejen z hlediska sémantického webu, je elektronický obchod. V této oblasti lze uvést přínos ontologie Good Relations pro jednotný popis produktů – tato aplikace je podrobněji rozvedena v kapitole 2.

Dalším oborem, kde ontologie nalézají uplatnění, je průmysl. Typickou aplikací sémantického webu se stává hledání. Hledání se ale nemusí omezovat na oblast webových dokumentů – podobný přístup se dá využít například pro data z návrhu výrobní linky (Obitko a kol., 2009). Schopnost data integrovat a schopnost formulovat složité strukturované dotazy se ukazuje jako velká výhoda, která byla dosažena prostou konverzí dat do jazyků RDF a OWL. Tato aplikace umožňuje pokládání a vyhodnocování dotazů, jako je například „najdi oblast kódu v tomto automatu, která mohla způsobit rozsvícení tohoto alarmu". Dalším příkladem je použití ontologií v multiagentních systémech. I jednoduchá skutečnost, že agenti nemají pevně zakódované reakce na zasílané zprávy, ale že jsou přizpůsobeni k zpracování zpráv zapsaných v ontologii, která se může během práce agenta rozšiřovat, znamená významný pokrok pro rozšiřitelnost a integraci. Multiagentní systémy se samozřejmě používají i jinde než v čistě průmyslových aplikacích. V oblasti průmyslu je příkladem systém pro řízení výroby, kde rozšíření multiagentího systému o nové komponenty či o schopnost vyrábět jiné výrobky je záležitostí úpravy znalostní báze nebo rozšíření ontologie (Obitko a kol., 2009), přičemž tato změna nevyžaduje ani zastavení systému. Obě tyto aplikace jsou podrobněji popsány v kapitole 5.

1.16 Závěr

Uvedený výklad pouze naznačuje některé otázky návrhu, implementace a využití ontologií. Jedním z cílů bylo ukázat, jak se problematika ontologií dotýká širokého spektra vědních oborů, které zahrnují filosofii, lingvistiku, logiku, umělou inteligenci, informatiku a jež se promítají do mnoha oblastí běžného života. Katalogy obchodních domů, programy, kterými si na internetu rezervujeme letenky, auta v půjčovách, místa v hotelích, kupujeme zboží, jsou aplikace, s nimiž se setkáváme nejčastěji. Kromě nich existují méně viditelné aplikace v řízení složitých systémů v oblasti technologie, samosprávy, zdravotnictví a dalších. Rozsáhlé ontologie a sémantické lexikony, jako Wordnet nebo Unified Medical Language System (UMLS), nejen slouží jako bohaté informační zdroje, ale lze je také použít k vývoji a integraci dílčích ontologií. Samostatnou a rychle rostoucí aplikací je sémantický web. Vytváření komplexních služeb sémantického webu se neobejde bez podpory ontologiemi.

Po tomto úvodním textu následují kapitoly, které vybrané aspekty popisují ve větších podrobnostech. Kapitola 2 se zabývá sémantickým webem, který je jednou z nejviditelnějších aplikací ontologií. Tato kapitola zejména popisuje jazyky RDF a OWL s ukázkami použití a s diskusí současných směrů. Kapitola 3 popisuje detailně uvažování v deskripčních logikách, které jsou použity jako základ pro jazyk OWL. Rozvíjí tak formálněji to, co je v předchozích kapitolách popsáno spíše intuitivně. Kapitola 4 se věnuje ontologickému inženýrství, tedy tomu, jak ontologie vytvářet. Důraz je kladen na jazyky používané na sémantickém webu a je ilustrován příklady, které ukazují, na co si dát při vytváření ontologií pozor. Kapitola 5 o aplikacích v průmyslové výrobě popisuje ve větších detailech aplikaci pro hledání ve strukturovaných datech a aplikaci v multiagentním systému. Kapitola 6 se věnuje sémantickému anotování a extrakci informací, což je nezbytné pro rozšíření ontologií do existujících systémů, včetně webu.

Součástí tohoto textu je poměrně rozsáhlý seznam literatury, který poskytuje ve většině souvisejících oborů výchozí bod pro další studium.

Literatura

Aristotelés: *Kategorie*. Praha 1958.

Baader F., Calvanese D., McGuinness D. L., Nardi D., Patel-Schneider P. F. (eds): *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

Brickley D., Guha R. V.: RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation*. http://www.w3.org/TR/rdf-schema/, 2004.

Carnap R: *Meaning and Necessity. A Study in Semantics and Modal Logic*. Phoenix Books, University of Chicago, 1947.

Cimiano P., Eberhart A., Hitzler P., Oberle D., Staab S., Studer R.: The smartweb foundational ontology. *SmartWeb Project Report*, 2004.

Date C. J.: *An Introduction to Database System*. 6th Edition. Addison-Wesley, 1995.

Ehrig M., Sure Y.: Ontology mapping – an integrated approach. In: *Proc. of the European Semantic Web Symposium (ESWS)*, 2004, s. 76–91.

Euzenat J., Shvaiko P.: *Ontology matching*. Springer, Berlín, Heidelberg, New York 1998.

Fagin R., Halpern J. Y., Moses Y., Vardi M. Y.: *Reasoning about Knowledge*. The MIT Press. Cambridge, Mass., 1995.

Fellbaum C. (ed.): *WordNet An Electrionic Lexical Database*. The MIT Press. Cambridge, Mass., 1998.

Frege G.: Über Sinn und Bedeutung. In: *Zeitschrift für Philosophie und philosophische Kritik*. 100, 1892, s. 25–50.

Genesereth M. R., Nilsson N. J.: *Logical Foundations of Articial Intelligence*. Morgan Kaufmann, Palo Alto 1987.

Genesereth M. R., Fikes R. E.: *Knowledge Interchange Format*, Version 3.0 Reference Manual. Computer Science Department, Stanford University, Technical Report Logic-92-1, June 1992.

Göckel R.: *Lexicon philosophicum*. Frankfurt a. M. 1613.

Goméz-Pérez A., Fernández-López M., Corcho O.: *Ontological Engineering*. Springer, Londýn 2004.

Gruber T. R.: Ontolingua: A Mechanism to Support Portable Ontologies. *KSL91-66*. Knowledge Systems Laboratory. Stanford University, 1992.

Gruber T. R.: A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2), 1993, s. 199–220.

Guarino N., Giaretta P.: Ontologies and Knowledge Bases. In: Mars N. J. I. (ed.): *Towards Very Large Knowledge Bases*. IOS Press, Amsterdam 1995.

Guarino N.: Understanding, building, and using ontologies: A commentary to „Using Explicit Ontologies in KBS Development", by van Heijst, Schreiber and Wielinga. *International Journal of Human and Computer Studies 46*, 1997, s. 293–310.

Guarino N.: Some Ontological Principles for Designing Upper Level Lexical Resources. *Proc. of the First International Conference on Lexical Resources and Evaluation*, Granada, 28–30 May 1998a.

Guarino N.: Formal Ontology and Information Systems. In: Guarino N. (ed): *Proceedings of FOIS'98*. IOS Press, Amsterdam, Trento 1998b, s. 3–15.

Guarino N., Welty C. A.: A Formal Ontology of Properties. In: Dieng R., Corby O. (eds): *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*. LNCS, vol. 1937. Springer-Verlag, Londýn 2000, s. 97–112.

Guarino N., Welty C.: Evaluating Ontological Decisions with OntoClean. *Communications of the ACM*, 45(2), 2002, s. 61–65.

Hennig W.: *Grundzüge einer Theorie der phylogenetischen Systematik*. Berlín 1950.

ISO 2788-1986. Documentation – Guidelines for the establishment and development of monolingual thesauri. *International Organisation for Standardization (ISO)*. Britský identický ekvivalent BS 5723, 1987.

ISO 21127-2006. Definition of the CIDOC Conceptual Reference Model http://www.iso.org, 2006.

Kershenbaum A., Fokoue A., Patel C., Welty C., Schonberg E., Cimino J., Ma L., Srinivas K., Schloss R., Murdock J W.: A View of OWL From the Field: Use-cases and Experiences. In: *Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions*. CEUR-WS. http://CEUR-WS.org/Vol-216/, 2006.

Klir J.: *Architecture of Systems Problem Solving*. Plenum Press. New York 1985.

Kolář P.: *Argumenty filosofické logiky*. Filosofia, Praha 1999.

Kripke S.: *Naming and Necessity*. Blackwell Publishing, 1981.

Lorhard J.: *Ogdoas scholastica*. St. Gallen 1606.

Matoušek K., Kouba Z.: Uncertain Temporal Support in Multiagent Medical Service. In: *Information Technology For Balanced Manufacturing Systems*. IFIP. Springer, Boston 2006, s.79–88.

Mařík V., Štěpánková O., Lažanský J. a kol.: *Umělá inteligence (1)*. Academia, Praha 1993.

Manola F., Miller E.: RDF Primer. *W3C Recommendation*, http://www.w3.org/TR/rdf-primer/, 2004.

Masolo C., Borgo S., Gangemi A., Guarino N., Oltramari A.: Ontology library. WonderWeb Deliverable no. 18. Online available from http://wonderweb.semanticweb.org/deliverables/D18.shtml, 2002.

Masolo C., Borgo S., Gangemi A., Guarino N., Oltramari A., Schneider L.: The WonderWeb Library of Foundational Ontologies (WFOL). *Technical report*, http://wonderweb.semanticweb.org/ deliverables/D17.shtml, 2002.

McBride B.: The Resource Description Framework (RDF) and its Vocabulary Description Language (RDFS). In: Staab S., Studer R. (eds) *Handbook on Ontologies*. 2004, s. 51–66.

McCarthy J.: Circumscription – A Form of Nonmonotonic Reasoning. *Artificial Intelligence 13*, 1980, s. 27–39.

McGuinness D. L., Harmelen van F.: *OWL Web Ontology Language Overview. W3C Recommendation*. http://www.w3.org/TR/owl-features/, 2004.

Miller G. A.: *Předmluva k WordNet*, 1998a.

Miller G. A.: Nouns in WordNet. In: Fellbaum C. (ed.): *WordNet An Electrionic Lexical Database*, The MIT Press. Cambridge, Mass., 1998b, s. 23–67

Motta E.: *Reusable components for knowledge modelling*. IOS Press, 1999.

Noy N. F., McGuinness D. L.: *Ontology Development 101: A Guide to Creating Your First Ontology*. Stanford University, Stanford 2001. http://www.ksl.stanford.edu/people/dlm/papers/ontology101/ontology101-noy-mcguinness.html

Obitko M., Vrba P., Mařík V., Radakovič, M.: The Impact of Semantic Technologies on Industrial Systems. In: *Handbook of Research on Social Dimensions of Semantic Technology and Web Services*. Hershey, IGI Global, Pennsylvania 2009, s. 759–779.

Oltramari A., Gangemi A., Guarino N., Masolo C.: Restructuring WordNet's top-level: the OntoClean approach. In: *Proceedings of OntoLex2002: Workshop on Ontologies and Lexical Resources*, Las Palmas 2002.

Øhrstrøm P., Andersen J., Schärfe H.: What Has Happened to Ontology. *Conceptual Structures: Common Semantics for Sharing Knowledge*, 2005, s. 425–438.

Reale G.: *Platón. Pokus o novou interpretaci*. Praha 2005.

Romanelli M., Buitelaar P., Engel R., Sonntag D., Reithinger N., Loos B., Porzel R., Zorn H.-P., Micelli V., Schmidt C., Weiten M., Burkhardt F., Zhou, J.: DOLCE ergo SUMO: On foundational and domain models in SWIntO (SmartWeb integrated ontology). *Technical report*, AIFB, University of Karlsruhe, 2006.

Russell B.: On Denoting, *Mind. 14*. 1995, s. 479–493.

Russell S., Norvig P.: *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

Schneider L.: How to Build a Foundational Ontology. The Object-Centered High-level Reference Ontology OCHRE. In: Guenther A., Kruse R., Neumann B. (eds): *KI 2003: Advances in Artificial Intelligence. Proceedings of the 26th German Conference on Artificial Intelligence (KI 2003)*. LNCS 2821. Springer, Heidelberg 2003, s. 120–134.

Schön D. A.: *The Reflective Practitioner.* Ashgate Publishing Ltd., Aldershot 2003.

Simon H. A.: The Structure of Ill Structured Problems, *Artificial Intelligence 4*, 1973, s. 181–201.

Smith B.: *Ontology and Informations Systems*. SUNY at Buffalo, 2004.

Social Care Online: Subject Taxonomy: The Social Care Institute for Excellence, http://www.scie.org.uk/, 2006, přístup 25. 2. 2008.

Sowa, J. F.: Conceptual Structures: Information Processing. In: *Mind and Machine*, Addison–Wesley, Reading, MA, 1984.

Sowa J. F.: *Knowledge Representation: Logical, Philosophical and Computational Foundations*. Brooks Cole Publishing Co, 2000.

Stanford Encyclopedia of Philosophy. http://plato.stanford.edu/, 2009, přístup 12. 3. 2009.

Strawson P. F.: *Individuals: An Essay in Descriptive Metaphysics.* Methuen, Londýn 1959.

Svátek V.: Ontologie a WWW. *DATAKON 2002.* Brno. 19–22. 10. 2002. s. 1–35.

Svobodová J., Svobodová D., Kuldanová P., Gejgušová I., Rosová M., Novák R.: *Čeština po síti. Katedra českého jazyka a literatury s didaktikou PdF OU.* http://www.osu.cz/fpd/kcd/dokumenty/cestinapositi/, 2003, přístup květen 2009.

Turing A. M.: Computing Machinery and Intelligence, *Mind 49*, 1950, s. 433–460.

Uschold M., Gruninger M.: Ontologies: principles, methods, and applications, *Knowledge Engineering Review*, 11(2), 1996, s. 93–155.

Vacura M.: Systém kategorií u Aristotela. In: *E-LOGOS*. Praha. http://e-logos.vse.cz, 2009, přístup 12. 3. 2010.

Wiggins D.: *Sameness and Substance Renewed.* Cambridge University Press, 2001.

Welty C., Guarino N.: Supporting ontological analysis of taxonomic relationships, *Data & Knowledge Engineering 39*. Elsevier, 2001, s. 51–74.

Zdrahal Z., Knoth P., Collins T., Mulholland P.: Reasoning across Multilingual Learning Resources in Human Genetics. *Conference ICL 2009*. September 23–25, Villach 2009.

2 Ontologie a sémantický web

V předchozí kapitole jsme uvedli sémantický web jako jednu z oblastí, která stála za rozmachem používání ontologií v počítačových aplikacích. Asi bychom nenašli jiné prostředí, které poskytuje tolik dat, má takový vliv na dnešní aplikace a zároveň je do určité míry heterogenní. Aplikace ontologií v tomto prostředí je určitě velkou výzvou, která jde nad rámec uzavřených systémů. Rozvoj v oblasti sémantického webu se přelévá i do ostatních oblastí – technologie sémantického webu se používají i v aplikacích, které mají s webem málo společného. Z těchto důvodů stojí za to se na tyto technologie podívat důkladněji. Výklad v této kapitole si neklade za cíl vyčerpávající formální popis současných možností, ale měl by být spíše prvním seznámením, aby byl čtenář schopen se zorientovat v technologiích pro reprezentaci znalostí na sémantickém webu, zejména s jazyky RDF, RDFS, OWL a SPARQL, včetně jejich konkrétního použití v rámci existujících webových standardů. Po uvedení do reprezentace znalostí se věnujeme zpracování znalostí na sémantickém webu, tj. hlavně vyhledávání a zpracování ontologií. Důležitou částí kapitoly je též popis současných výrazných aplikací a trendů na sémantickém webu.

2.1 Sémantický web

Současný WWW se dá popsat jako knihovna dokumentů propojených odkazy. Technologie běžně požívané na WWW umožňují přenos dokumentů (nejčastěji ve formátu HTML) a jejich prezentaci uživatelům. Je to tedy místo, kde lidé pracují a počítače poskytují pouze přenos dokumentů a jejich prezentaci.

Představme si, že si chceme na dnešním webu zařídit dovolenou, ale nechceme využít služeb cestovní kanceláře. Máme svou představu, jak by měla dovolená probíhat, takže se posadíme k počítači a hledáme vhodnou lokalitu dle našich preferencí, v dané lokalitě ověříme možnost vhodného ubytování, možnost dopravy do dané lokality, ověříme reference atd. Pokud něco v tomto procesu nevyhovuje, hledáme dále, dokud nejsme spokojeni. Ve chvíli, kdy jsme spokojeni, uskutečníme rezervaci ubytování a dopravy a zaplatíme. Celou dobu jsme nedělali nic jiného, než že jsme používali počítač k tomu, aby nám ukazoval dokumenty, tyto dokumenty jsme četli a na jejich základě se rozhodovali a prováděli další akce. Dovedeme si ale určitě představit, že by toto vše za nás provedl počítač, včetně toho, že by nám byl schopen své akce na požádání vysvětlit a zdůvodnit – my bychom mu jen dodali naše omezení a preference. Vzhledem k tomu, že velká část současných webových stránek je generovaná z databází, nemusel by při vhodném formátu pro přenos informací být problém ani se zpracováním přirozeného jazyka. Výhodu podobného asistenta oceníme zejména pro práci v doméně, které příliš nerozumíme, například vyhledání nějakého zboží dle zadaných parametrů.

Sémantický web (SW) je zamýšlen jako rozšíření současného webu, v němž počítače navíc budou rozumět obsahu uložených a přenášených informací a budou tedy schopny

pracovat autonomněji. Budou tedy například schopny provést úkol popsaný v předchozím textu. Krátká výstižná definice SW dle práce (Berners-Lee, Hendler, Lassila, 2001) zní: „Sémantický web je rozšířením současného webu, v němž je informacím dán dobře definovaný význam a který lépe umožňuje spolupracovat počítačům a lidem". Zastřešující organizací aktivit souvisejících se sémantickým webem je přirozeně World Wide Web Consortium (W3C). Dle tohoto konsorcia je sémantický web definován takto: „Sémantický web je reprezentace dat na WWW. Je to společné úsilí vedené W3C s velkým počtem účastníků z výzkumných a průmyslových partnerů. SW je založen na Resource Description Framework (RDF), který integruje různé aplikace za použití XML pro syntax a URI pro pojmenování." V následujícím výkladu si popíšeme, co vlastně tato definice znamená, co vše a jak by mělo realizaci sémantického webu umožnit.

2.2 Reprezentace znalostí na sémantickém webu

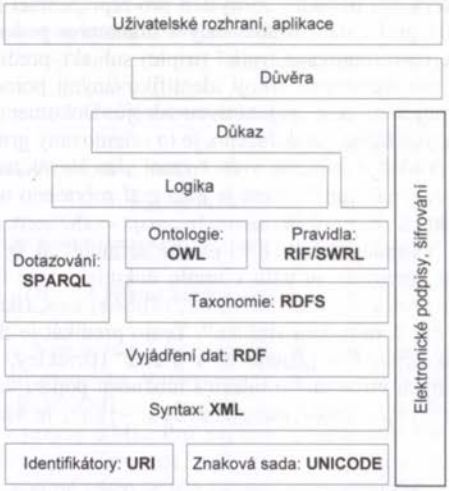
Současný web, stejně jako sémantický web, se skládá z takzvaných zdrojů (resource). Za zdroj můžeme obecně považovat cokoliv, čemu lze přiřadit vlastní identitu. Zdroj konceptuálně vznikne přiřazením označení (identifikace) k této identitě. Protože tento vznik je konceptuální, zdroj nemusí být konstantní, tj. může se měnit v čase, a dokonce nemusí v daném čase vůbec existovat. Klasickým příkladem zdroje v kontextu webu je stránka v jazyce HTML umístěná na webovém serveru, která může být jednoznačně identifikována pomocí svého URL. Jiným příkladem zdroje dle této definice je například obrázek ve formátu PNG, postscriptový dokument na ftp serveru, budova identifikovaná svou adresou, zboží identifikované čárovým kódem nebo kódem RFID, kniha identifikovaná svým kódem ISBN nebo slovo v knize identifikované číslem stránky, řádkem a pozicí na řádku.

2.2.1 Architektura sémantického webu

Aby byla možná komunikace mezi různě schopnými agenty pracujícími na sémantickém webu, je nutná interoperabilita v několika vrstvách. Určitě je třeba sdílet používanou syntax a sémantiku, ale k tomu přistupují další požadavky, jako například požadavek důvěry ve vyměňovanou informaci. Zamýšlená architektura sémantického webu se dá nastínit pomocí vrstev, viz obr. 2.1.

Popišme si vrstvy odspodu tohoto schématu – nejprve ve shrnující podobě, konkrétní příklady pro vybrané vrstvy budou uvedeny hned poté. Ve spodních dvou vrstvách je definována společná syntax pro identifikaci a reprezentaci zdrojů. Jako znaková sada se používá Unicode, což dává možnost zahrnout do řetězců znaky z různých národních abeced. Identifikaci zdrojů zajišťuje URI (Uniform Resource Identifier), což je řetězec s definovanou syntaxí, který může být použit pro jednoznačnou identifikaci. Příkladem URI je URL (Uniform Ressource Locator) zajišťující adresování zdrojů na WWW – například `http://www.example.org/~jan/` je jednoznačná adresa popisující umístění nějakého zdroje, jež se dá také chápat jako jednoznačný identifikátor tohoto zdroje (dokumentů). Pro popis dokumentů na syntaktické úrovni se často, i když ne výlučně, používá XML (Extensible Markup Language), včetně jmenných prostorů (pro rozlišení zdroje značek), a XML Schema (pro definici struktury dokumentu, zde využitelné zvláště pro popis literálů).

Po definici syntaxe přichází na řadu sémantika. Základní konstrukty pro vyjadřování tvrzení o zdrojích poskytuje jazyk RDF (Resource Description Framework). Tento jazyk byl primárně zamýšlen pro vyjadřování metadat o webových zdrojích, jakými jsou například jméno, autor, či datum vytvoření konkrétní webové stránky, nicméně je použitelný na jakékoliv zdroje, nejen na webové. Jazyk fakticky popisuje informace pomocí orientovaného grafu. V RDF můžeme používat libovolné identifikátory pro popis zdrojů. Abychom mohli dát těmto identifikátorům význam ve smyslu ontologie, je třeba přidat jejich popis. Přinejmenším potřebujeme souhrn použitých identifikátorů, tj. slovník. Základní zavedení slovníku a případně jeho popis ve formě zejména taxonomie tříd a vlastností umožňuje jazyk RDFS (RDF Schema). Tento jazyk je dále rozšířen jazykem OWL (Web Ontology Language) pro bohatší ontologické konstrukty. Oba tyto jazyky jsou nezávislé na doméně. Dá se říci, že RDF je jazyk pro vyjádření znalostní báze, RDFS je jazyk pro popis slovníku a OWL je jazyk pro vyjádření ontologie. Ne všechny znalosti se dají vyjádřit pomocí OWL – pro vyjádření pravidlových znalostí jsou navrhovány jazyky RIF a SWRL, které umožňují popisovat Hornovy klauzule.



Obr. 2.1 Architektura sémantického webu.

Pro dotazování nad informacemi vyjádřenými v RDF se používá jazyk SPARQL (Protocol and RDF Query Language). Protože RDFS, OWL, SWRL, RIF atd. jsou vyjádřené v RDF, lze SPARQL používat i pro dotazování nad informacemi vyjádřenými v těchto jazycích.

Jak již bylo uvedeno dříve, pro využití ontologií je důležité uvažování. V dalších patrech obrázku je tedy uvedena logika spolu s důkazem. Jazyk OWL je založen na deskripční logice, kde je zaručené konzistentní, úplné a rozhodnutelné uvažování (o deskripčních logikách byla krátká zmínka v předchozí kapitole a do hloubky je rozvedena v kapitole 3). Můžeme se tedy spolehnout, že pokud jeden agent z informací na sémantickém webu něco odvodí, bude to konzistentní s tím, co odvodí ze stejných zdrojů

jiný agent. Máme též zaručeno, že v konečném čase oba agenti mohou odvodit všechny důsledky a že tyto důsledky budou stejné za použití stejného inferenčního mechanismu. Nicméně pro důvěru ve výsledek uvažování musíme mít též důvěru ve zdroje, tedy v premisy uvažování. Důvěryhodnost zdrojů je na obrázku zajištěna pomocí elektronických podpisů a dalších kryptografických prostředků.

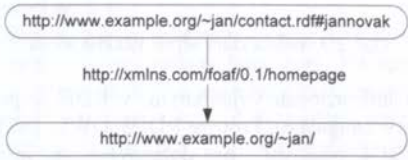
Nad tímto vším mohou pracovat aplikace tak, jak je uvedeno ve vizích sémantického webu – aplikace budou schopny si nalézt na webu na základě zadání uživatele potřebné informace, provést rozhodnutí a akce ke splnění požadované úlohy.

Rozveďme nyní popis vybraných technologií z obrázku s cílem poskytnout první přiblížení jejich hlavních rysů, ovšem bez nároku na úplnost. V případě zájmu lze představu doplnit z doporučení W3C, kterými se řídí i implementace či nástroje, jež s těmito technologiemi pracují.

2.2.2 RDF – Resource Description Framework

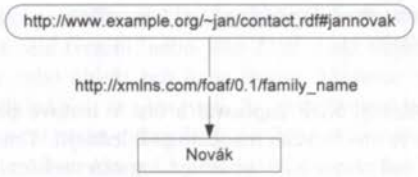
Jazyk RDF slouží pro reprezentaci dat či informací o zdrojích (resource) v grafové podobě. Protože tento jazyk byl původně zamýšlen pro reprezentaci metadat o webových zdrojích, je orientovaný graf s daty tvořen uzly a hranami v podobě webových zdrojů s URI. Každá hrana je reprezentována trojicí (triple) subjekt–predikát–objekt. Všechny elementy této trojice jsou webovými zdroji identifikovanými pomocí URI. Dá se tedy také říci, že trojice je pojmenované spojení dvou zdrojů. Dokument v RDF je pak množinou těchto orientovaných trojic, jinak řečeno, je to orientovaný graf.

Jako nejjednodušší příklad si můžeme uvést tvrzení „Jan Novák má domovskou stránku `http://www.example.org/~jan/`", které je jako graf zobrazeno na obr. 2.2. První element v trojici, subjekt `http://www.example.org/~jan/contact.rdf#jannovak`, popisuje Jana Nováka. Všimněme si, že URI přesně definuje, jak se k dokumentu dostat, a také jak se dostat ke konkrétnímu uzlu v tomto dokumentu ve formátu RDF. Druhý element, predikát `http://xmlns.com/foaf/0.1/homepage`, říká, že mezi subjektem a objektem je relace „má domovskou stránku". Tento predikát je definován pomocí jazyka RDFS ve slovníku FOAF – „friend of a friend" (Brickley, Miller, 2010), který slouží pro popis vybraných informací o lidech s možností popisu vazeb mezi lidmi. Poslední element, objekt `http://www.example.org/~jan/`, je vlastní domácí stránka Jana Nováka.



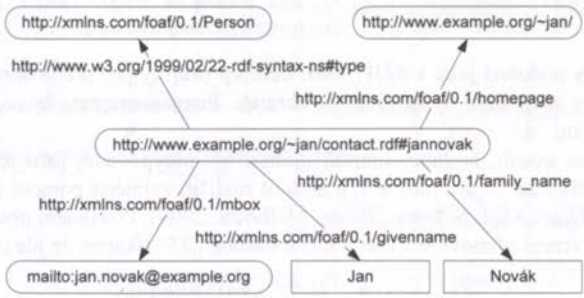
Obr. 2.2 Příklad trojice v RDF v podobě grafu.

Uvedli jsme, že všemi elementy trojice musí být zdroje ve formě URI. V případě posledního elementu, objektu, však platí výjimka – objektem může být též literál, například tedy řetězec, číslo celé či reálné, datum, booleovská hodnota atd. Hodnota může být netypovaná nebo typovaná pomocí datových typů jazyka XML Schema. Jako příklad uveďme tvrzení „Jan Novák má příjmení Novák" zobrazené v podobě grafu na obr. 2.3.



Obr 2.3 Trojice v RDF s literálem v podobě grafu.

Kombinací více takovýchto trojic dohromady vznikne zajímavější graf – viz obr. 2.4. Tento graf vyjadřuje, že Jan Novák má uvedenou domovskou stránku, že má příjmení Novák a křestní jméno Jan, že má uvedený e-mail, a také že je instancí třídy „Osoba". Toto poslední tvrzení je vyjádřeno pomocí predikátu `type`, který je definován přímo v základním slovníku RDF.



Obr. 2.4 Příklad komplexnějšího grafu v RDF.

Grafická podoba je sice názorná pro člověka, ale pro počítačové vyjádření, přenos a sdílení takovýchto informací se používají jiné formáty. Normativní formát pro vyjádření RDF se nazývá RDF/XML a v souladu s architekturou sémantického webu (obr. 2.1) je to dokument v jazyce XML. Pro zajištění syntaktické interoperability by každý nástroj či systém, který s RDF pracuje, měl umět zpracovávat hlavně tento normativní formát. Existují samozřejmě nástroje pro editaci, které informace v RDF umožňují zobrazit a editovat v podobě přijatelnější pro člověka – uživatel tento formát určený výhradně pro počítače vůbec nemusí znát. Z tohoto důvodu a spíše jen pro ilustraci uveďme, jak by vypadala serializace informací uvedených na obr. 2.4 ve formátu RDF/XML:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    xmlns="http://www.example.org/~jan/contact.rdf#">
  <foaf:Person rdf:about=
        "http://www.example.org/~jan/contact.rdf#jannovak">
    <foaf:mbox rdf:resource="mailto:jan.novak@example.org"/>
    <foaf:homepage
          rdf:resource="http://www.example.org/~jan/"/>
    <foaf:family_name>Novák</foaf:family_name>
```

```
    <foaf:givenname>Jan</foaf:givenname>
  </foaf:Person>
</rdf:RDF>
```

Mezi těmi, kdo potřebují RDF zapisovat a číst v textové podobě, si získal oblibu formát TURTLE, který je pro člověka mnohem přehlednější. Tento formát je podmnožinou formátu N3, který nad rámec RDF obsahuje logická rozšíření a pravidla. Serializace informací z obr. 2.4 v těchto jazycích vypadá takto:

```
@prefix :      <http://www.example.org/~joe/contact.rdf#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

:jannovak a foaf:Person ;
      foaf:givenname "Jan" ;
      foaf:family_name "Novák" ;
      foaf:homepage <http://www.example.org/~jan/> ;
      foaf:mbox <mailto:jan.novak@example.org> .
```

První řádky podobně jako v RDF/XML definují prefixy pro přehlednost, další řádky zapisují trojice za použití syntaktických zkratek. Poznamenejme, že `a` je syntaktická zkratka predikátu `rdf:type`.

Jak již jsme uvedli, hodnoty literálů mohou být netypované, jako je tomu ostatně v uvedeném příkladu. Je ale možné typ hodnot rozlišit, zejména pomocí typů definovaných ve specifikaci XML Schema (Biron, Malhotra, 2004). Příkladem použití typovaného literálu ve tvrzení „Janovi je 35 let", kde o literálu „35" říkáme, že jde o celé číslo, je

```
Jan máVěk "35"^^xsd:integer.
```

V případě složitějších konstrukcí, jako jsou například seznamy, nemusíme nutně chtít, aby byl každý uzel identifikován pomocí URI. Pokud nám stačí vyjádřit existenci uzlu, na který se chceme odkazovat jen v rámci jednoho dokumentu (tedy ne zvnějšku), můžeme použít takzvaný *anonymní uzel*, označovaný také jako *b-node* (blank node). Anonymní uzel může být v RDF použit jen na místě subjektu nebo objektu, ale ne na místě predikátu. Příklad použití anonymního uzlu je ukázán na obr. 2.5. Z logického hlediska anonymní uzel říká „existuje zdroj takový, že ...".

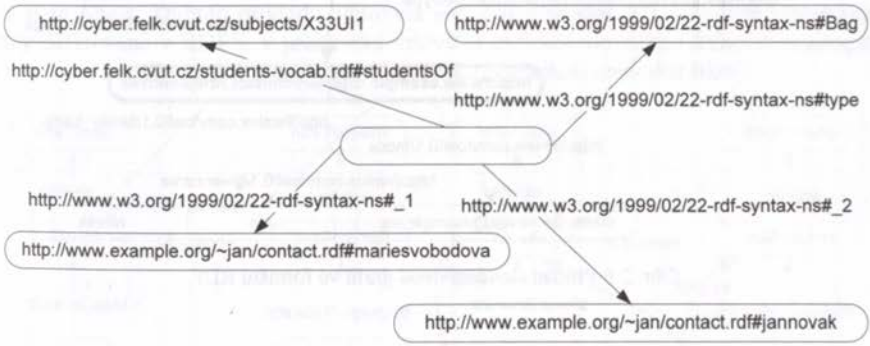
Dalším konstruktem, který tento obrázek ukazuje, je definice skupiny – seznamu studentů zapsaných na předmět. Skupiny v RDF mohou být tohoto typu:

- kupa (rdf:Bag) je skupina, která může obsahovat své prvky (tj. zdroje nebo literály) několikrát, přičemž nezáleží na pořadí,
- sekvence (rdf:Seq) je skupina, která může obsahovat své prvky několikrát, přičemž záleží na pořadí,
- alternative (rdf:Alt) je skupina, která obsahuje různé alternativy, většinou hodnoty jedné vlastnosti (například slovo v různých jazycích).

Tyto uvedené skupiny neuzavírají výčet prvků. Jinak řečeno, neříkají, že k již specifikovaným prvkům nelze již nic přidat. Pro vyjádření toho, že výčet je již úplný, se v RDF používají takzvané kolekce. Jejich konstrukce je podobná jako v jazycích Prolog nebo Lisp – kolekce je tvořená uspořádaným seznamem prvků a je uzavřená prvkem, který reprezentuje prázdný seznam.

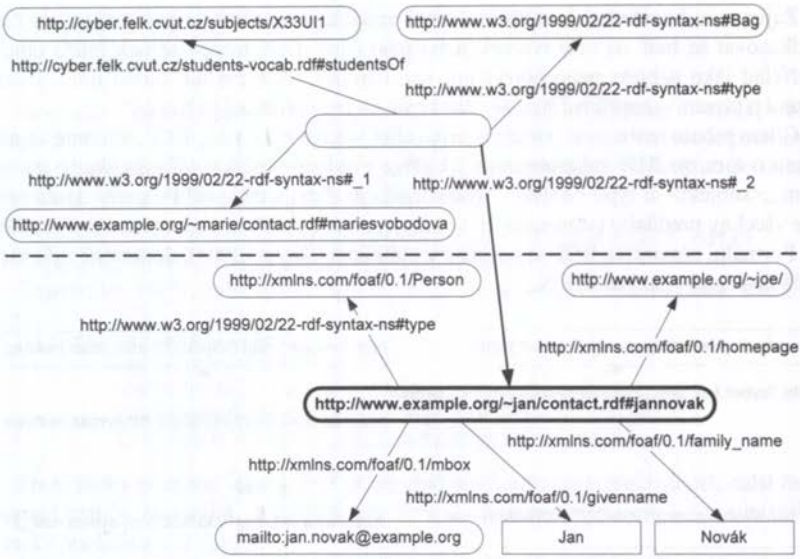
Zajímavou konstrukcí je reifikace, která umožňuje rozebrat tvrzení (trojici) na části a odkazovat se buď na celé tvrzení, nebo jeho části. Celá trojice se pak může objevit například jako subjekt nebo objekt pro jinou trojici. Můžeme tak mimo jiné vytvářet tvrzení o tvrzení – například říci, že „Jan tvrdí, že jeho příjmení je Novák".

Cílem tohoto textu není podat vyčerpávající výklad o formátu RDF, zmiňme se tedy už jen o tom, že RDF také obsahuje již dříve uvedený predikát rdf:type, který má význam „<subjekt> je typu <objekt>" (kde objekt je třída) a třídu rdf:Property, která zahrnuje všechny predikáty (vlastnosti, tj. ty zdroje, které se mohou objevit na místě predikátu). Formální sémantika RDF je popsána v publikaci (Hayes, 2004), dodatečné upřesnění je též dostupné v jazyce RDFS.



Obr. 2.5 Příklad anonymního uzlu a reprezentace seznamu v RDF.

Vraťme se ještě k základu RDF – použití URI jako identifikátorů zdrojů. URI zasazují RDF do prostředí webu, protože poskytují nejen jednoznačnou identifikaci, ale též určují, jak se k danému zdroji dostat. Pokud tedy v rámci nějakého dokumentu ve formátu RDF použijeme zdroj mimo tento dokument, můžeme pomocí WWW tento zdroj nalézt a získat o něm další informace. Například při pohledu na obrázky 2.2 nebo 2.4 můžeme na adrese `http://xmlns.com/foaf/0.1/homepage` nalézt další informace k tomuto predikátu. Můžeme tak jednoduše dohledávat upřesňující či vysvětlující informace a tím také být schopni další informace odvozovat. Stojí ale za zmínku, že vzhledem k tomu, že se pohybujeme v distribuovaném a neřízeném prostředí webu, se může stát, že některý zdroj je nedostupný stejně, jako jsou dnes nedostupné některé webové stránky. Systémy, které se sémantickým webem pracují, se musí s tímto faktem vyrovnat. Jedna úloha URI v RDF je tedy zasazení tohoto jazyka do prostředí WWW, a tak tedy tvořit *sémantický web*. Druhá související a neméně důležitá úloha je jednoznačná identifikace zdrojů, která je potřebná pro integraci dat z různých dokumentů. Pokud dostaneme dva dokumenty, které obsahují stejné URI, můžeme tyto zdroje sloučit dohromady – tak, jak je to znázorněno na obr. 2.6. Slučování je samozřejmě možné jen pro zdroje s URI – pro anonymní uzly platí, že jsou dostupné jen lokálně v daném dokumentu, a tedy nemohou být sloučeny s anonymními uzly jiného dokumentu (alespoň na zde popsané úrovni, tj. v RDF bez dalších pravidel).



Obr. 2.6 Příklad sloučení dvou grafů ve formátu RDF.

2.2.3 RDFS – RDF Schema

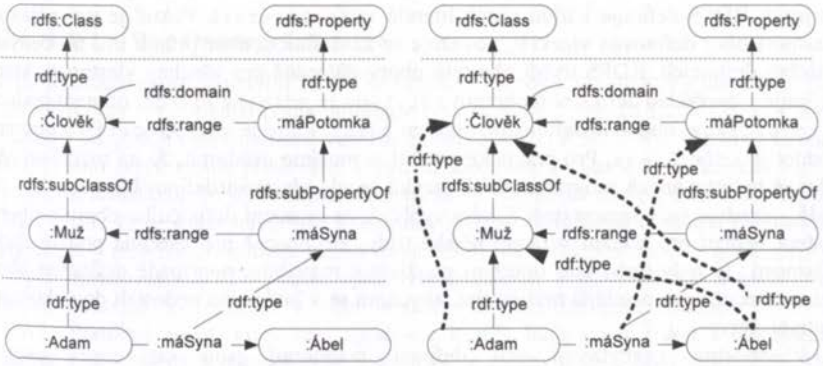
Jak můžeme vidět, RDF je velmi obecný jazyk pro reprezentaci dat. Pro sdílení a využití těchto dat je ale třeba vymezit význam použitých zdrojů identifikovaných pomocí URI. Již jsme uvedli, že páteří ontologie je často taxonomie. Taxonomické vztahy mezi třídami a vlastnostmi (predikáty) umožňuje vyjádřit právě jazyk RDFS – RDF Schema. Název RDF Schema je udržován z historického hlediska, správnější popis významu jazyka RDFS, který je uveden i v jeho specifikaci (Brickley, Guha, 2004), je „jazykem pro popis slovníku RDF". RDFS nám tedy umožňuje definovat termíny a jejich omezení zejména ve tvaru taxonomií, které již lze považovat za ontologie, alespoň velmi základní. RDFS také dodefinovává vztahy mezi termíny definovanými v RDF.

Před dalším přiblížením jazyka uveďme jednoduchý příklad použití RDFS, ke kterému se můžeme vracet v dalším obecném popisu:

```
@prefix :      <http://www.example.org/sample.rdfs#> .
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

:Clovek    a rdfs:Class.
:Muz       rdfs:subClassOf :Clovek.

:maPotomka a rdfs:Property;
           rdfs:range :Clovek;
           rdfs:domain :Clovek.
:maSyna    rdfs:subPropertyOf :maPotomka;
```

```
                rdfs:range :Muz.

:Adam           a :Muz;
                :maSyna :Abel.
```

V prvních řádcích po definici prefixů definujeme třídy a jejich taxonomii – říkáme, že `Člověk` je třída (připomeňme, že `a` je zkratka pro `rdf:type`), která má podtřídu `Muž`. Dále definujeme vlastnosti `máPotomka` jako vlastnost (relaci, predikát), která jako definiční obor (domain) a obor hodnot (range) má třídu `Člověk`. Vlastnost `máSyna` je pak podvlastností vlastnosti `máPotomka` s tím, že jeho obor hodnot se zužuje na instance třídy `Muž`. Tolik tedy definice slovníku a omezení pomocí jazyka RDFS, v posledních řádcích je ilustrováno již jen použití tohoto slovníku, kde říkáme, že Adam je muž, který má syna Ábela. Tomuto příkladu odpovídá levá polovina obr. 2.7 – v horní vrstvě jsou třídy definované v RDFS, v prostřední vrstvě je definice slovníku (definice ontologie) a ve spodní vrstvě je ilustrováno použití slovníku (vyjádření znalostní báze).



Obr. 2.7 Příklad definice slovníku v RDFS a jeho použití, včetně ukázky odvozených faktů.

Pojďme se nyní podívat na jazyk RDFS podrobněji. RDFS definuje `rdfs:Resource` jako třídu všech zdrojů popsatelných v RDF. Všechny ostatní třídy jsou tedy podtřídou `rdfs:Resource`. Všechny zdroje, které jsou třídami, jsou instancí `rdfs:Class`. Znamená to tedy, že `rdfs:Class` je též instancí `rdfs:Class` a že `rdfs:Resource` je instancí `rdfs:Class`. Poznamenejme, že definice tříd je konceptuální, a tedy stejná množina instancí může být popsána různými třídami. Například třída lidí, kteří mají e-mail `jan.novak@example.org`, je jiná než třída lidí, kteří mají domovskou stránku `http://www.example.org/~jan/`. Ale extense těchto tříd, tedy množina instancí, které do těchto tříd spadají, je stejná – je to jednoprvková množina obsahující zdroj `http://www.example.org/~jan/contact.rdf#jannovak`, tedy Jan Novák z obr. 2.5. RDFS nám umožňuje říci, že některé zdroje slouží pro popis tříd a že jiné zdroje jsou instancemi daných tříd. Množina tříd a množina instancí však nejsou disjunktní – může se stát, že instancí třídy je také (jiná) třída. Jinak řečeno, lze popsat metatřídy, tedy třídy, jejíž instance jsou opět třídami.

Vlastnosti nám umožňují definovat vztahy mezi zdroji, popř. mezi zdrojem a literálem. V RDFS je třída všech vlastností označená `rdf:Property`. Vztahy mezi již uvedenými třídami mohou být na první pohled matoucí – protože `rdf:Property` je třída, je `rdf:Property` instancí třídy `rdfs:Class`; protože je to třída zdrojů, je též podtřídou třídy `rdfs:Resource`; protože je to zdroj, je též instancí třídy `rdfs:Resource`. Jazyky RDF a RDFS se takto snaží klást minimální omezení na možný popis dat a jejich struktury a umožňují tím velkou flexibilitu (ovšem ne vždy takovou flexibilitu chceme, viz odst. 2.2.4).

Vraťme se ale zpět k vlastnostem, tedy zdrojům, které mohou v RDF trojici vystupovat v podobě predikátu. RDFS zařazuje do `rdf:Property` i vlastnost z RDF, kterou jsme již uvedli – `rdf:type`. Tato vlastnost nám popisuje relaci mezi instancí třídy a třídou. Každá vlastnost může mít definován definiční obor (`rdfs:domain`) a obor hodnot (`rdfs:range`). Definiční obor je třída instancí, které se mohou v trojici, kde na místě predikátu je daná vlastnost, objevit na místě subjektu. Oproti tomu obor hodnot je třída instancí, které se v takovéto trojici mohou objevit na místě objektu (poznamenejme, že jazyk RDFS definuje i třídu všech literálů `rdfs:Literal`). Pokud je pro nějakou vlastnost obor definován vícekrát, považuje se za definiční obor průnik tříd uvedených v těchto definicích. RDFS uvádí oba tyto obory důsledně pro všechny vlastnosti, které definuje – například definičním oborem `rdf:type` je `rdfs:Resource`, oborem hodnot je `rdfs:Class`; obdobně definičním oborem `rdfs:range` je `rdf:Property` a oborem hodnot je `rdfs:Class`. Pro praktické použití si musíme uvědomit, že na rozdíl od objektově orientovaných programovacích jazyků je zde vlastnost definována nezávisle na třídě – existuje tedy samostatně. Z toho vyplývá, že omezení definičního oboru a oboru hodnot neplatí pro použití v rámci nějaké třídy, ale obecně pro všechna použití dané vlastnosti. Je tedy třeba tato omezení používat s rozvahou, popřípadě definovat obor velmi obecně i pro pozdější rozšiřování, abychom se v budoucnu nedostali do nekonzistentního stavu.

Vlastnostmi nezbytnými pro definici taxonomií jsou `rdfs:subClassOf` a `rdfs:subPropertyOf`. Použití obou bylo ilustrováno v úvodním příkladu – první vlastnost slouží pro definici podtříd, druhá vlastnost pro definici podvlastností. V případě „`:Muz rdfs:subClassOf :Clovek`" říkáme, že každá instance třídy `Muž` je též instancí třídy `Člověk`. V případě „`:maSyna rdfs:subPropertyOf :maPotomka`" říkáme, že pokud je mezi dvěma zdroji vztah `maSyna`, pak je mezi nimi i vztah `maPotomka`. Obě vlastnosti jsou z definice tranzitivní.

RDFS ještě definuje několik dalších tříd a vlastností, které však již nejsou tolik podstatné pro úvodní přiblížení. Příkladem je vlastnost `rdfs:label`, která umožňuje přiřadit zdroji jméno použitelné pro člověka, což může být důležité pro uživatelská rozhraní pracující nad daty vyjádřenými v RDF.

Důležitá je možnost inference, tedy odvozování nových faktů, na základě ontologie definované pomocí RDFS. V pravé polovině obr. 2.7 jsou jako příklad znázorněna některá fakta, která lze odvodit v rámci znalostní báze na základě definice ontologie. Odvozování může probíhat na základě odvozovacích pravidel (entailment rules) a axiomů uvedených v práci (Hayes, 2004). Příkladem axiomu je „`rdfs:range rdfs:domain rdf:Property`" (můžeme to číst jako „vlastnost `rdfs:range` je definována pouze pro vlastnosti"). Axiomy jsou tedy tvrzení, která definují podstatu jazyka a která se při

odvozování přidají ke grafu, v němž chceme odvozování provázet. Odvozovací pravidla mají část, která říká, co v grafu hledat, a část, která říká, co v případě nalezení do grafu přidat. Příklad pravidla je: *[rdfs9] pokud graf obsahuje* „`u rdfs:subClassOf x`" *a* „`v rdf:type u`", *pak přidej* „`v rdf:type x`". Na základě tohoto pravidla a informace o tom, že `Adam` je `Muž` a že `Muž` je podtřídou třídy `Člověk` (viz obr. 2.7), lze odvodit informaci, že `Adam` je `Člověk`. Podobně dle pravidla *[rdfs3] pokud graf obsahuje* „`p rdfs:range x`" *a* „`u p v`", *pak přidej* „`v rdf:type x`" a informace o tom, že Adam má syna Ábela, lze odvodit, že `Ábel` je `Muž`. Podobným způsobem lze odvodit i další fakta uvedená na obr. 2.7. Na základě aplikace těchto pravidel lze spočítat deduktivní tranzitivní uzávěr, tedy množinu (či graf) faktů, která můžeme odvodit z grafu obsahujícího znalostní bázi a ontologii.

Na první pohled je toto uvažování velmi jednoduché. Nicméně i takto málo sémantiky znamená velké možnosti pro aplikace, zejména v distribuovaném prostředí, kde je třeba zpracovávat informace z různých zdrojů. Velkou výhodou je to, že informace pro toto uvažování jsou obsaženy v ontologii.

2.2.4 OWL – Web Ontology Language

Popis ontologie prostřednictvím RDFS se omezuje na popis taxonomie s některými dodatečnými rysy. Pro bohatší popis ontologie (viz kapitola 1) slouží jazyk OWL – Web Ontology Language (McGuinness, van Harmelen, 2004), (W3C OWL Working Group, 2009). Tento jazyk slouží především k využití možností a vlastností deskripční logiky v prostředí sémantického webu. Zjednodušeně se dá říci, že jde o zápis deskripční logiky v RDF s tím, že se snaží stavět nad konstrukty RDFS. Tomuto postupnému rozšiřování výraziva je v architektuře sémantického webu věnována velká pozornost. Agent, který je schopen zpracovat jen RDF, může zpracovat dokument v jazyce OWL a porozumět alespoň částečně informacím v něm obsaženým, přestože není schopen uvažovat v OWL. Podobně agent, který umí zpracovat RDFS, bude umět získat z dokumentu v jazyce OWL alespoň taxonomické informace a využít je pro další uvažování. Deskripční logika použitá za základ poslední verze jazyka OWL je *SROIQ* (Motik a kol., 2009).

Nicméně použitá deskripční logika neumožňuje takovou volnost jako RDF nebo RDFS, například není možné používat metatřídy. Toto vede k tomu, že OWL staví na RDFS a RDF, ale omezuje volnost těchto jazyků. OWL se snaží o kompatibilitu tím, že staví na výrazových prostředcích RDF a RDFS tak, aby význam dokumentu v OWL byl co nejbližší jeho interpretaci v RDF a RDFS. Striktně řečeno, tato kompatibilita je pouze syntaktická – syntakticky validní dokument v OWL je též vždy syntakticky validním dokumentem v RDF a RDFS. Obráceně to obecně neplatí, neboť validní dokument v RDFS nemusí být validním dokumentem v OWL ve smyslu použité deskripční logiky. Pro některé aplikace ale může být užitečné používat konstrukce z jazyka OWL, přestože si výsledek neklade za cíl být použitelný i při zpracování pomocí deskripční logiky. Z tohoto důvodu se zavádí několik úrovní či profilů jazyka OWL, aby bylo zřejmé zamýšlené použití dané ontologie, a také aby bylo možné určit úroveň a náročnost podpory uvažování.

Vrátíme se k nim později, podívejme se nejprve na jazyk OWL v podobě ilustrativního popisu s příklady. Bude se vlastně jednat i o popis použité deskripční logiky s tím,

jak je tato logika zasazena do RDF a jaké používá identifikátory. Hned na začátku uveďme, že to, co z terminologie deskripční logiky známe jako koncept, je v OWL označováno jako třída, a to, co z deskripční logiky známe jako roli, je v OWL označováno jako vlastnost (property).

Ontologie a znalostní báze v OWL může obsahovat zejména tyto oblasti:
- definici tříd, taxonomické a jiné relace mezi třídami;
- objektové a datové vlastnosti, ale na rozdíl od RDFS se rozlišuje, zda je obor hodnot vlastnosti zdrojem nebo literálem; toto zahrnuje charakteristiky těchto vlastností;
- individua, která jsou obvykle instancemi tříd, podobně jako u RDFS je individuum definováno nezávisle a může být klasifikováno jako instance třídy nebo tříd, ale příslušnost ke třídě nemusí být explicitně vyjádřena.

Před dalším výkladem si ukažme příklad malé ontologie popisující pizzy (doména je inspirována publikací (Horridge, 2009)), nejprve v syntaxi deskripční logiky:

$$\texttt{Pizza} \sqsubseteq \exists \texttt{máKorpus}.\texttt{Korpus}$$
$$\texttt{Pizza} \sqcap \texttt{Korpus} \equiv \bot$$
$$\texttt{NevegetariánskáPizza} \equiv \texttt{Pizza} \sqcap \neg\ \texttt{VegetariánskáPizza}$$
$$\texttt{Tr}(\texttt{jeIngrediencíPro})$$
$$\texttt{jeIngrediencíPro} \equiv \texttt{máIngredienci}^{-}$$

Syntax a sémantika deskripčních logik je popsána detailněji v kapitole 3, zde si vystačíme s ilustrativním popisem – je dobré si uvědomit i tímto způsobem, na čem je jazyk OWL založen. Popis ontologie sestává z množiny axiomů, tedy tvrzení, která jsou daná a která vymezují, co v ontologii platí a co z ní lze odvodit.

První řádek říká, že `Pizza` je něco, co má korpus, definujeme tak tedy (částečně) třídu `Pizza`. V definici dále figuruje vlastnost `máKorpus` a třída `Korpus`, přičemž výraz `∃máKorpus.Korpus` vymezuje třídu všech věcí, které participují ve vztahu `máKorpus` tím způsobem, že existuje alespoň jedna participace, kde hodnota vlastnosti je věc ze třídy `Korpus`. Celý řádek říká to, že `Pizza` je podtřídou tohoto výrazu, jinak řečeno, být pizzou má v naší ontologii za nutnou (nikoliv postačující) podmínku mít korpus. Na druhou stranu zde ale netvrdíme, že cokoliv, co má korpus, musí také být nutně pizzou.

Druhý řádek říká, že třídy `Pizza` a `Korpus` nemají společné žádné instance – jejich průnikem je prázdná množina (v OWL označovaná jako `owl:Nothing`, viz další text). Tímto axiomem říkáme pouze to, že pokud je něco `Pizza`, nemůže to už být `Korpus` a naopak. Takováto tvrzení jsou velmi důležitá pro automatické uvažování, více o tomto tématu pojednávají kapitoly 3 a 4.

Třetí řádek definuje třídu `NevegetariánskáPizza` jako něco, co je sice `Pizza`, ale ne vegetariánská (k definici vegetariánské pizzy se vrátíme později). V tomto případě, na rozdíl od prvního řádku, používáme ekvivalenci tříd – říkáme tedy, že tato podmínka je jak nutná, tak postačující. Při automatickém uvažování tak můžeme vše, co splňuje uvedenou definici, zařadit do třídy `NevegetariánskáPizza` – pro první řádek toto neplatí.

Poslední dva řádky uvádějí příklad axiomů popisujících vlastnosti. Předposlední řádek říká, že vlastnost `jeIngrediencíPro` je tranzitivní – pokud je mouka ingrediencí korpusu a korpus je ingrediencí pizzy, pak i mouka je ingrediencí pizzy. Poslední řádek definuje (opět ve smyslu nutné a postačující podmínky), že vlastnost `jeIngrediencíPro` je vlastností inverzní k `máIngredienci`, a tedy platí-li, že nějaká pizza má jako

ingredienci nějaký korpus, pak určitě platí i to, že tento korpus je ingrediencí dotyčné pizzy (vzájemná inverznost těchto vlastností samozřejmě platí i pro jiné objekty než pizzy a korpusy).

Pro úplnost jen uveďme, že syntax deskripční logiky je dána historicky, může tedy na první pohled působit nezvykle, a že sémantiku lze definovat také prostřednictvím predikátové logiky (připomeňme, že deskripční logika je výpočetně lépe zvládnutelná podmnožina predikátové logiky): třídy odpovídají unárním predikátům, vlastnosti binárním predikátům, individua konstantám. Pro další informace o deskripčních logikách odkazujeme čtenáře na kapitolu 3, pro popis sémantiky OWL pomocí rozšíření deskripční logiky *SROIQ* na práci (Motik a kol., 2009) a pro popis sémantiky OWL pomocí rozšíření sémantiky RDF na práci (Schneider, 2009).

Pro popis ontologie v OWL v počítačovější podobě se používají syntaxe strukturně-funkční nebo manchesterská. Strukturně-funkční syntax (Motik a kol., 2009) je vhodná pro formální zápis struktury ontologií. Popis uvedené ontologie v této syntaxi je následující:

```
Prefix(p:=<http://example.com/pizza.owl#>)
Ontology(<http://example.com/pizza.owl>
  SubClassOf(p:Pizza ObjectSomeValuesFrom(p:máKorpus p:Korpus))
  DisjointClasses(p:Pizza p:Korpus)
  EquivalentClasses(p:NevegetariánskáPizza
     ObjectIntersectio-
nOf(ObjectComplementOf(p:VegetariánskáPizza) p:Pizza))
  TransitiveObjectProperty(p:jeIngrediencíPro)
  InverseObjectProperties(p:jeIngrediencíPro p:máIngredienci)
)
```

Manchesterská syntax (Horridge a Patel-Schneider, 2009) je používána v různých editorech a dalších nástrojích, z určitého pohledu může být přehlednější. Je též používána v kapitole 4 pojednávající o ontologickém inženýrství. Uvedená ontologie v této syntaxi vypadá takto:

```
Prefix: p: <http://example.com/pizza.owl#>
Ontology: <http://example.com/pizza.owl>
Class: p:Pizza
    SubClassOf:
        p:máKorpus some p:Korpus
    DisjointWith:
        p:Korpus
Class: p:NevegetariánskáPizza
    EquivalentTo:
        p:Pizza
         and (not (p:VegetariánskáPizza))
ObjectProperty: p:jeIngrediencíPro
    Characteristics:
        Transitive
    InverseOf:
        p:máIngredienci
```

Normativní zápis ontologie v OWL je ale pouze ve formátu RDF/XML (viz popis serializace RDF), vnoření OWL do RDF je popsáno v práci (Patel-Schneider a Motik,

2009). Tento formát je vyžadován pro interoperabilitu nástrojů, které s jazykem OWL pracují. Čtení a zápis v ostatních formátech jsou možné, často mohou být užitečnými, ale nejsou povinné. V případě jazyka OWL ještě více platí naše předchozí tvrzení – že formát RDF/XML je pouze pro počítače, člověk by se neměl snažit ontologie v tomto jazyce vytvářet.

Jen pro představu si jako ilustrační příklad uveďme, jak by zhruba vypadal zápis prvního řádku naší malé ontologie vnořený do jazyka RDF, kde definujeme anonymní uzel typu `owl:Restriction`, který vyjadřuje dané omezení pro třídu `Pizza`:

```
p:Pizza rdfs:subClassOf
  [ a owl:Restriction;
    owl:onProperty p:máKorpus;
    owl:someValuesFrom p:Korpus].
```

Jak vidíme, zápis v manchesterské nebo strukturně-funkční syntaxi bude většinou srozumitelnější. Editory pro vytváření ontologií v jazyce OWL, které samozřejmě umí ontologii zapsat nejen v syntaxi RDF/XML, ale i dalších formátech, uvedeme v dalším výkladu. Vraťme se nyní k popisu vlastního jazyka OWL.

2.2.4.1 Třídy a individua

Individua reprezentují objekty v doméně, kterou popisujeme. V případě naší ontologie je to například konkrétní pizza (s konkrétními přísadami atd.). Třídy v OWL popisují množiny individuí – třída tedy může být interpretována jako nějaká podmnožina množiny všech individuí. Třídy jsou formálně definovány prostředky deskripční logiky použité v OWL.

Základem pro popis ontologie je často taxonomie, tedy hierarchická klasifikace, která definuje mezi třídami vztah – *je podtřídou*. Tato relace subsumpce označuje jedním směrem specializaci (zúžení), druhým směrem generalizaci (zobecnění). Typickým příkladem je biologická Linného taxonomie. V naivní podobě můžeme říci, že `Kočka` je podtřídou třídy `Živočich`, což znamená, že vše, co je `Kočka`, je zároveň `Živočich` a na druhou stranu, aby něco mohlo být `Kočka`, musí to být také `Živočich`. Dalším podobným příkladem z naší ontologie je fakt, že `VegetariánskáPizza` je podtřídou třídy `Pizza`, což se dá zapsat explicitně jako

```
VegetariánskáPizza rdfs:subclassOf Pizza.
```

Zde je důležité si uvědomit rozdíl mezi OWL (jazyce založeném na deskripční logice) a třeba objektově orientovaným programováním, jak ho známe například z jazyků C++ nebo Java. V objektově orientovaných jazycích tuto hierarchii tříd definujeme s tím, že je to jediný prostředek, jak vztah *je podtřídou* vyjádřit. Objekty jsou pak definovány jako objekty konkrétní třídy s tím, že jsou zároveň objekty tříd, které jsou *z těchto definic* obecnější. Podtřída je specifikována především touto definicí – přidání atributů nebo specializace implementace metod není samo o sobě důvodem k onomu vztahu mezi třídami.

Oproti tomu ontologie je vytvářena především s úmyslem automatického uvažování, včetně automatické klasifikace tříd i individuí. Když v naivní ontologii vyjádříme, že `Pes` je něco, co má čtyři nohy, a k tomu přidáme, že `Dalmatin` má čtyři nohy a bílou

srst s černými skvrnami, pak vztah, že `Dalmatin` je podtřídou třídy `Pes` nemusíme explicitně uvádět – může být odvozen automaticky. Jen pro úplnost dodejme, že zatímco hierarchie tříd v objektově orientovaných jazycích je obecně omezena na strom, v OWL žádné takové omezení není.

Podobně je tomu s instancemi tříd. To, že nějaké individuum je instancí třídy v OWL, můžeme vyjádřit explicitně (pomocí `rdf:type`, viz uvedený popis RDF), podobně jako když v programovacím jazyce řekneme, že vytváříme objekt jakožto instanci třídy. Pokud ale víme, že individuum má čtyři nohy a srst bílou s černými skvrnami, pak zařazení do tříd `Dalmatin` a `Pes` proběhne automaticky. Individuum může být popsáno, aniž bychom explicitně poznamenali cokoliv o jeho příslušnosti k nějaké třídě.

Významnými třídami jsou v OWL `owl:Thing` a `owl:Nothing`. První z těchto tříd („věc") zahrnuje všechna individua, a tudíž všechny třídy jsou zároveň podtřídou třídy `owl:Thing`. Oproti tomu `owl:Nothing` („nic") je interpretována jako prázdná množina a odpovídá nesplnitelným (nekonzistentním) omezením. Pro kompatibilitu s RDFS[1] jsou všechny třídy v OWL instancemi třídy `owl:Class`, která je podtřídou třídy `rdfs:Class`.

Mimo subsumpce můžeme mezi třídami vyjádřit disjunkci, tedy to, že neexistuje individuum, které by bylo instancí obou tříd. Takovéto tvrzení jsme již viděli dříve jako vztah mezi třídami `Korpus` a `Pizza`. Podobně bude platit disjunkce mezi třídami `Pizza` a `Přísada` a stejně tak mezi třídami `Korpus` a `Přísada`. Pokud takovéto axiomy z ontologie vynecháme, přijdeme obyčejně o důležitou informaci, která nám pak může chybět při automatickém uvažování. Obvykle (ale ne vždy!) toto platí v taxonomiích – různé specializace jedné třídy by sice měly znamenat také přidání takových dodatečných omezení, aby byla zamýšlená disjunkce těchto specializací zřejmá, ale pokud z nějakého důvodu nechceme provádět modelování domény do takových podrobností, měli bychom přidat alespoň axiomy o disjunkci. Pokud tedy zavedeme třídy `MasováPřísada`, `ZeleninováPřísada` a `SýrováPřísada` jako podtřídy třídy `Přísada` bez další specifikace, měli bychom také dodat axiom o disjunkci těchto tříd. Když pak budeme mít libovolnou zeleninovou přísadu, budeme moci odvodit, že takováto přísada nemůže být zároveň masová.

V OWL lze též říci, že dvě třídy nebo dvě individua jsou shodná nebo rozdílná. Je důležité podotknout, že v OWL neplatí, že různé názvy znamenají různé entity. Během automatického uvažování můžeme odvodit, že rozdílně pojmenovaná individua reprezentují stejnou entitu. Dokud explicitně nevyjádříme, nebo během uvažování neodvodíme, že by různě pojmenované entity byly stejné nebo rozdílné, jsou možné oba stavy.

2.2.4.2 *Objektové a datové vlastnosti*

Jak již jsme uvedli, vlastnosti v OWL vyjadřují relace. Jazyk OWL rozlišuje dva typy vlastností – objektové a datové. Objektové vlastnosti jsou relace mezi dvěma individui, zatímco datové vlastnosti jsou relace mezi individuem a literálem, jako je například číslo.

Zaměřme se nejprve na objektové vlastnosti. Příkladem takovéto vlastnosti z naší ontologie o pizzách je `jeIngrediencíPro`. Podobně jako u tříd lze mezi vlastnostmi

[1] Připomeňme, že tato kompatibilita znamená to, že agent, který rozumí jen jazyku RDFS, může také přečíst nějaké užitečné informace v dokumentu zapsaném v OWL. Ne vše z těchto rysů lze popsat v deskripční logice, jež slouží zejména pro syntaktickou kompatibilitu.

pomocí `rdfs:subPropertyOf` definovat hierarchické vztahy. V naší ontologii tak může platit, že `jeKorpusemPro` a `jePřísadouPro` jsou specializace vlastnosti `jeIngrediencíPro`. Tyto vlastnosti můžeme využít třeba v následujícím vztahu mezi individui (při použití ontologie pro zachycení konkrétního stavu světa v naší doméně):

```
šunkováPřísada1 jePřísadouPro mojePizza.
```

Mezi vlastnostmi dále můžeme vyjádřit další vztahy, užitečná je například možnost vyjádřit inverzní vlastnosti. Můžeme třeba říci, že `máPřísadu` je vlastností inverzní k vlastnosti `jePřísadouPro`. Pro náš příklad bychom pak z uvedeného popisu světa při automatickém uvažování odvodili, že také platí

```
šunkováPřísada1 jeIngrediencíPro mojePizza.
mojePizza máPřísadu šunkováPřísada1.
```

Podobně jako u tříd můžeme k vlastnosti navíc přiřadit její ekvivalentní a disjunktní vlastnosti. Mimo to lze vlastnosti charakterizovat jako relace – jazyk OWL umožňuje vyjádřit, že objektová vlastnost je funkční, inverzně funkční, tranzitivní, symetrická, asymetrická, reflexivní nebo antireflexivní. Vlastnost samozřejmě může mít více těchto charakteristik.

Již jsme si uvedli, že vlastnost `jeIngrediencíPro` je tranzitivní, a tedy pokud je šunka ingrediencí pro šunkovou přísadu, pak v uvedeném případě je ingrediencí i pro mou pizzu. Vlastnost může být dále charakterizována jako symetrická. Příkladem takové vlastnosti je `máSourozence` – pokud platí, že Adam má sourozence Bořka, pak i Bořek má sourozence Adama. Symetrická vlastnost je také sama o sobě inverzní vlastností. Oproti tomu pro asymetrickou vlastnost `jeDítětem` platí, že pokud Bořek je dítětem Davida, tak nemůže platit, že David je dítětem (stejného) Bořka. Samozřejmě to ale nevylučuje, že David je dítětem někoho jiného. Pro funkční vlastnosti platí, že pro každé individuum může být nejvýše jedno individuum, které participuje na této vlastnosti. Příkladem funkční vlastnosti je `máManžela`. Kdybychom ve znalostní bázi našli tvrzení, že Alena má manžela Jana a Honzu, museli bychom usoudit, že jde o stejné individuum (již jsme se zmínili, že v OWL neplatí, že různá jména musí automaticky znamenat rozdílné entity). Pokud by bylo explicitně uvedeno, že Jan a Honza je někdo jiný, dospěli bychom k nekonzistenci. Inverzně funkční vlastnost je pak taková, jejíž inverzní vlastnost je funkční – příkladem je vlastnost `jeManželemPro`, a v obdobném příkladu s Janem a Honzou s touto vlastností bychom dospěli ke stejným závěrům. Reflexivní vlastnost je taková, kde tato vlastnost musí mít individuum samo se sebou. Příkladem takovéto vlastnosti je `zná` – člověk zná sám sebe, ale to samozřejmě nevylučuje to, že může znát i někoho jiného. Oproti tomu antireflexivní vlastnost je taková, která dává do vztahu individua, která nemohou být stejná. Příkladem takovéto vlastnosti je již zmíněná vlastnost `jeDítětem` – nikdo nemůže být sám sobě dítětem.

Jak již bylo uvedeno u popisu jazyka RDFS, k vlastnostem lze specifikovat definiční obor (`rdfs:domain`) a obor hodnot (`rdfs:range`). Například pro vlastnost `máManžela` můžeme říci, že její definiční obor odpovídá třídě `Žena` a že její obor hodnot je třída `Muž`. Pak z tvrzení `Alena máManžela Jan` lze odvodit, že `Alena` je `Žena` a že `Jan` je `Muž`.

Mimo popsané objektové vlastnosti umožňuje jazyk OWL používat i datové vlastnosti. Datové vlastnosti mají vždy jako obor hodnot datový typ. Příkladem jsou řetězce, čísla, datum atd. Hodnoty mohou být netypované, je ale možné typ hodnot rozlišit pomocí typů definovaných ve specifikaci XML Schema (Biron, Malhotra, 2004), jak jsme již viděli v popisu jazyka RDF.

Vidíme, že použití datových vlastností je podobné jako u objektových vlastností. Tyto dva typy vlastností jsou samozřejmě neslučitelné – každá vlastnost je buď datová, nebo objektová. I pro datovou vlastnost můžeme specifikovat následující upřesnění: její ekvivalentní a disjunktní vlastnosti, její definiční obor a obor hodnot a dále je možné vlastnost charakterizovat jako funkční. Je zřejmé, že datová vlastnost vzhledem k jejímu oboru hodnot (literály, ale ne individua) nemůže být například symetrická.

Podobně jako u vztahu příslušnosti individua do třídy platí, že pokud explicitně nebo pomocí automatického uvažování neodvodíme, že relace mezi dvěma individui existuje nebo neexistuje, je možné cokoliv. Viděli jsme možnost explicitně vyjádřit fakt, že relace existuje. Stejně tak v OWL můžeme pro objektovou i datovou vlastnost vyjádřit fakt, že taková relace neexistuje (pomocí `NegativeObjectPropertyAssertion` a `NegativeDataPropertyAssertion`).

Zejména pro databázové aplikace je důležité, že v jazyce OWL můžeme deklarovat klíče, tedy vyjádřit fakt, že instance nějaké třídy jsou jednoznačně identifikované hodnotou určité vlastnosti. Například člověk může být v ontologii identifikován rodným číslem. Za zmínku též stojí, že vlastnosti můžeme skládat, a tím definovat nové vlastnosti, například vlastnost `máStrýce` lze definovat řetězcem `máRodiče` – `máBratra`.

2.2.4.3 *Modelování tříd*

Vraťme se k popisu tříd v jazyce OWL s tím, že nyní pro tento popis budeme používat i vlastnosti. Třídu individuí můžeme popsat především tak, že specifikujeme vlastnosti, ve kterých může participovat. Příkladem je třída $\exists$ `máDítě.`$\top$, což je množina všech, kdo mají nějaké dítě, nebo třída $\exists$`máKorpus.Korpus`, což je třída všeho, co má `Korpus`. Tato omezení mohou nabývat různých tvarů – příkladem jsou omezení založená na kvantifikátorech a omezení kardinality.

Začněme s omezeními založenými na kvantifikátorech. Existenčnímu kvantifikátoru odpovídají příklady uvedené v předchozím příkladu – pro třídu $\exists$`máDítě.`$\top$ platí, že individua, která do této třídy patří, se v daném vztahu takto vyskytují alespoň jednou. Oproti tomu univerzálnímu kvantifikátoru odpovídá příklad třídy

$$\forall \texttt{máPřísadu.ZeleninováPřísada},$$

kde říkáme, že pro všechna individua z této třídy, která mají nějakou přísadu, platí, že tato přísada je zeleninová. Povšimněme si, že tímto neříkáme, že *všechna* individua musí mít aspoň jednu zeleninovou přísadu, ale pouze to, že přísada, pokud nějaká je, nemůže být jiná. Podobně je asi zřejmé, že v případě omezení $\exists$`máPřísadu.ZeleninováPřísada` platí, že tím definujeme třídu, jejíž instance mají *alespoň jednu* zeleninovou přísadu (tj. mohou mít i jiné přísady). Pro popis třídy individuí, které mají zeleninové přísady (tj. alespoň jednu takovou přísadu a žádné jiné), bychom mohli použít například kombinaci

$$\exists \texttt{máPřísadu.ZeleninováPřísada} \sqcap \forall \texttt{máPřísadu.ZeleninováPřísada}.$$

Jak jsme již viděli, hodnot jedné vlastnosti můžeme mít pro jedno individuum více – například více přísad. Pro případ, že bychom chtěli počet hodnot určité vlastnosti nějak omezit, slouží omezení kardinality. Kardinalitu můžeme stanovit maximální, minimální nebo ji vymezit přesně (což je shodné se současným stanovením minimální a maximální kardinality se stejnou hodnotou). Například pro definici třídy, která zahrnuje pizzy s nejméně třemi přísadami, použijeme zápis (omezujeme kardinalitu vlastnosti `máPřísadu` na minimálně tři hodnoty):

```
ZajímaváPizza ≡ Pizza ⊓ ⩾3 máPřísadu.
```

V uvedeném příkladu zajímavé pizzy ale není nijak omezeno, o jaké hodnoty jde. Jazyk OWL umožňuje i kvalifikované omezení kardinality, kde lze mimo počet hodnot omezit i to, z jaké třídy hodnoty jsou. Lze tak například definovat pizzu, která má právě tři *sýrové* přísady. Ve všech omezeních kardinality se počet použití počítá pro sémanticky různé hodnoty. Pokud například v bázi znalostí o pizze, která má mít právě tři přísady, najdeme informaci o čtyřech přísadách, ale dvě z těchto přísad jsou sémanticky ekvivalentní (tj. jsou to ekvivalentní individua), pak není toto omezení porušeno. Podobné případy mohou nastat zejména v distribuované bázi znalostí – pokud například jeden dokument na webu přiřazuje dané instanci pizzy dvě přísady a jiný dokument jiné dvě přísady, může se stát, že dvě z těchto přísad lze při automatickém uvažování označit za sémanticky shodné. To samozřejmě může nastat i v rámci jedné báze znalostí.

Další možností definice třídy je pomocí specifikace, které konkrétní hodnoty (tj. individua) musí obsahovat vztah za použití dané vlastnosti. Například italskou pizzu lze definovat tímto způsobem:

```
ItalskáPizza ≡ Pizza ⊓ máPůvod:Itálie.
```

Mimo vymezení třídy dle používaných vlastností je též možné třídu definovat vyjmenováním všech jejích instancí. Například třídu kontinent lze definovat vyjmenováním všech známých kontinentů, tj. individuí jako je `Evropa`, `Asie` atd.

Uvedenými způsoby definujeme omezení, která vymezují možné instance třídy – deklarujeme tak tedy (nepojmenovanou) třídu. Jak jsme se již zmínili dříve v malé ontologii popisující pizzy, tato omezení mohou být použita pro definici (pojmenovaných) tříd jako podmínky nutné nebo nutné a postačující. V jazyce OWL je možné použít pro definici třídy množinu takovýchto podmínek, přičemž se pak předpokládá platnost všech podmínek v množině. Rozlišují se množiny podmínek nutných a podmínek nutných a postačujících, přičemž je navíc možnost k definici třídy přiřadit více takovýchto oddělených množin. Třídy lze též definovat pomocí průniku nebo sjednocení jiných tříd.

2.2.4.4 Vlastnosti pro anotování

Ontologie v OWL potřebujeme pro možnost sémantického anotování dokumentů na webu (k tomu se vrátíme v dalším textu). Nicméně i samotné ontologie a jejich části je třeba anotovat, tedy dodávat k nim poznámky a další popisky pro umožnění lepšího porozumění jejich významu. Z tohoto důvodu OWL zavádí i vlastnosti použitelné pro anotování. Příkladem anotace je tvrzení `:Pizza dc:creator :Jan`, tedy že „třídu Pizza vytvořil Jan" nebo tvrzení `:PizzaOnto2 owl:priorVersion :PizzaOnto1`, tedy uvedení předchozí verze ontologie.

Jazyk OWL definuje anotační vlastnosti pro ontologie k označení předchozí verze ontologie a k označení nekompatibilit. Dále jsou definovány vlastnosti pro třídy, vlastnosti a individua, které umožňují přiřadit informaci o verzi, přiřadit název čitelný pro člověka (včetně názvů v různých jazycích), přidat komentář, odkaz na relevantní nebo další definující entity ontologie. Tyto vlastnosti slouží jen jako anotace pro člověka, nejsou použity při automatickém uvažování.

Mimo definované vlastnosti si můžeme definovat i vlastní anotační vlastnosti – koneckonců v uvedeném příkladu vidíme použití vlastnosti `dc:creator` ze slovníku Dublin Core[2]. Libovolná datová nebo objektová vlastnost může být označena za anotační. K těmto vlastnostem se navíc váží jistá omezení plynoucí z jejich zamýšleného použití: pro jejich hodnoty nelze použít anonymní třídy (anonymní třídy ale v OWL 2 anotovat lze), nemůžeme je používat pro definice tříd atd.

2.2.4.5 Importování ontologií a předpoklad otevřeného světa

Z předchozího oddílu víme, že ontologie se vytvářejí i proto, aby mohly být použity vícekrát, a to včetně jejich použití v jiných ontologiích. Toto platí v prostředí webu ještě více než v jiných aplikacích – ontologie potřebujeme nejen používat, ale i využívat pro další rozšiřování. Proto jazyk OWL zavádí možnost uvést import jiné ontologie. V ontologii se pak můžeme na entity importované ontologií odkazovat, dále je rozšiřovat, popř. zpřesňovat. Toto přispívá i k možnosti navrhovat ontologie v oddělených modulech pro jednodušší vývoj a udržování.

V souvislosti s tím je ale třeba si uvědomit předpoklady v jazyce OWL, jejichž dopad se importováním ontologií ještě zesiluje. Jedním takovým předpokladem, který se může lišit od našich zkušeností z jiných oblastí, je předpoklad otevřeného světa. Tento předpoklad říká, že pravdivost tvrzení je nezávislá na tom, jestli pozorovatel může potvrdit, zda je tvrzení pravdivé. Naproti tomu předpoklad uzavřenosti světa říká, že cokoliv, o čem nemůžeme na základě našich omezených znalostí zjistit, že by bylo pravdou, pravdou není. Tento předpoklad známe například z databází a je použitelný zejména tam, kde můžeme pro účely aplikace předpokládat, že máme všechny znalosti – co v databázi nenajdeme, to neplatí. V prostředí webu ale těžko můžeme dojít ke stavu, kde bychom měli všechny znalosti – a tedy o pravdivosti toho, o čem nevíme a co nemůžeme odvodit, nemůžeme pronášet žádné závěry. Jinak řečeno, pokud nejsme schopni ověřit pravdivost, neznamená to ještě, že je tvrzení nepravdivé.

Již jsme uvedli příklady toho, že tvrzení o tom, že něco neplatí, musíme uvést explicitně – nestačí pouze tvrzení neuvést. S tím souvisí i již uvedená nejedinečnost jmen. Pokud se například dvě třídy jmenují jinak, neznamená to ještě, že jsou rozdílné. Naopak, obecně dokud neprokážeme něco jiného, nelze o jejich vztahu nic říci – mohou být zcela odlišné, mohou mít neprázdný průnik, ale také mohou být zcela ekvivalentní. Je potřeba si uvědomit, že právě omezování toho, co je možné, je úkolem ontologie. Ve světě s prázdnou ontologií je možné vše, přidáváním axiomů do ontologie svět omezujeme, a tím vymezujeme, co je možné a nemožné. Ontologie je užitečná právě tím, že nám možné stavy světa omezí. Dále tím, že stále předpokládáme neúplnou informaci, umožňujeme monotónní uvažování – nemůže se nám tedy stát, že přidanou informací učiníme neplatným něco, co jsme odvodili dříve (pokud samozřejmě nepřidáme infor-

[2] Jednoduchý slovník (http://dublincore.org/documents/dces/) deklarující metadata pro popis zdrojů.

maci, která stojí proti nějakému existujícímu tvrzení). Na druhou stranu, není vždy možné nalézt odpověď zejména na otázky osahující negaci, například „nalézt lidi, které Jan nezná".

V souvislosti s možným rozšiřováním ontologií musíme být opatrní s přidáváním omezení. Pokud přidáme omezení, která se nám sice „hodí pro aplikaci", ale nejsou obecně platná, můžeme omezit opětovné použití ontologie. Typickým příkladem jsou omezení definičních oborů a oborů hodnot pro vlastnosti. Například, pokud jsme pro naši aplikaci uvedli, že doménou vlastnosti `máIngredienci` je `Pizza`, pak nemůžeme tuto vlastnost jednoduše použít v jiné podobné doméně. Pokud bychom tuto ontologii importovali do ontologie popisující zmrzliny a uvedenou vlastnost převzali, pak pro tvrzení `mojeZmrzlina máIngredienci Čokoláda` bychom museli odvodit, že `mojeZmrzlina` patří do třídy `Pizza`. Dodatečné uvedení axiomu, že oborem hodnot vlastnosti `máIngredienci` je `Zmrzlina`, by nám nepomohlo – řekli bychom tak spolu s axiomy z importované ontologie, že oborem hodnot je průnik tříd `Pizza` a `Zmrzlina`, což určitě není to, co bychom chtěli. Více o těchto i dalších předpokladech a zdánlivých paradoxech pojednává následující oddíl o ontologickém inženýrství. Zájemcům doporučujeme též článek (Rector a kol., 2004) s přehledem nejběžnějších omylů při používání jazyka OWL.

Poznamenejme, že i když bychom se měli snažit uvedené zásady dodržovat, na otevřeném webu zcela jistě dojde[3] k tomu, že bude existovat plno špatně navržených ontologií, které se nedají rozšiřovat, popř. jsou již od začátku nekonzistentní. Zrovna tak může dojít k tomu, že ontologie, kterou importujeme, z odkazovaného místa zmizí. Tyto problémy norma jazyka OWL neřeší, ale řešení situací, jako je nekonzistence a nedostupnost, je stále předmětem výzkumu.

2.2.4.6 Úrovně a profily jazyka OWL

Zmínili jsme se, že v jazyce RDF můžeme použít konstrukty, které v jazyce OWL nemají interpretaci. Z tohoto důvodu je jazyk OWL[4] rozdělen do dvou oblastí – OWL 2 DL a OWL 2 Full. OWL 2 DL je jazyk, který odpovídá našemu předchozímu popisu – deskripční logice s bohatším výrazivem zapsané v RDF. V OWL 2 DL máme garantované vlastnosti, jako je rozhodnutelnost a úplnost. Oproti tomu jazyk OWL 2 Full tvoří konstrukty OWL 2 DL s tím, že pouze rozšiřujeme jazyk RDFS s ponecháním syntaktické volnosti RDF. Pro jazyk OWL 2 Full není samozřejmě garantována žádná výpočetní vlastnost. Výběr konkrétního jazyka záleží na zamýšleném použití – zda potřebujeme více vyjadřovací volnosti nebo garanci výpočetních vlastností.

I jazyk OWL 2 DL ale může být obecně příliš komplexní pro určité aplikace. Z toho důvodu jsou zavedeny takzvané profily jazyka OWL, které se liší svou složitostí. Norma (W3C OWL Working Group, 2009) se zmiňuje o těchto třech profilech:

OWL 2 EL (dle rodiny deskripčních logik EL) je podmnožina jazyka OWL 2, která se hodí pro popis ontologií, kde je velký počet tříd a vlastností. Typickou aplikační oblastí je biomedicína, popisy produktů apod. Důležité pro tyto aplikace je to, že úlohy typu konzistence ontologie, subsumpce a ověření, zde je individuum instancí třídy, mohou být rozhodnuty v polynomickém čase.

---

[3] Či spíše již v hojné míře dochází.

[4] V celé kapitole hovoříme o současné verzi OWL 2, rozdělení je jiné než u předchozí verze tohoto jazyka, kde se rozlišovaly úrovně Lite, DL a Full.

Druhou podmnožinou jazyka OWL 2, OWL 2 QL (Query Language), je jazyk zaměřený na aplikace, které podobně jako relační databáze pracují nad velkým objemem instancí, ale nepožadují jejich složitý popis. Nejčastější úlohou v aplikacích tohoto jazyka je odpovídání na dotazy (dotazovací jazyk SPARQL je popsán v dalším výkladu). I když jde o relativně omezený jazyk vzhledem k OWL 2 Full, umožňuje tento jazyk používat hlavní rysy diagramů UML a ER. Je tedy předurčen pro aplikace, které požadují rozšíření možností relačních databází.

Poslední profil, OWL 2 RL (Rule Language), je jazyk s velkými vyjadřovacími schopnostmi při zachování škálovatelného uvažování. Tento jazyk je zamýšlen pro bohatý popis dat v RDF, s důrazem na možnost přidávání pravidel, je tedy vhodný pro logické programování s využitím deskripční logiky.

2.2.5 Dotazovací jazyk SPARQL

V předchozím textu jsme se věnovali především reprezentaci ontologií a dat. Pro použití v aplikacích ale potřebujeme dotazování. Pro jazyk RDF se používá dotazovací jazyk SPARQL (Prud'hommeaux, Seaborne, 2008), který má syntax na první pohled podobnou jazyku SQL, ovšem s tím, že umožňuje hledání v grafech popsaných v RDF. Vyhledávaný vzor je tedy specifikován především trojicemi.

Dotaz na nalezení e-mailů a domovských stránek všech Janů Nováků (viz obr. 2.4) může v jazyce SPARQL vypadat takto:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?email ?homepage WHERE {
    ?person a foaf:Person.
    ?person foaf:family_name "Novák".
    ?person foaf:givenname "Jan".
    ?person foaf:mbox ?email.
    ?person foaf:homepage ?homepage.
}
```

Vidíme, že syntax je podobná syntaxi jazyka TURTLE, v jazyce SPARQL je možné používat i stejné syntaktické zkratky. První řádek definuje prefix, druhý řádek říká, na co se ptáme, a další řádky popisují vzor, na který se ptáme. Proměnné začínají znakem otazník ?, vzor vyjadřuje trojice, které v grafu popsaném pomocí RDF hledáme. Odpovědí na dotaz je tabulka hodnot navázaných na jednotlivé proměnné tak, aby vyhověly hledaným vzorům. Máme-li data z obr. 2.4, bude výsledkem pouze jedna položka:

?email	?homepage
<mailto:jan.novak@example.org>	<http://www.example.org/~jan/>

V případě vyhledávání na webu, tedy pokud dotaz pošleme serveru, který je schopen dotaz zpracovat a odpovědět na něj (takzvaný „SPARQL endpoint" pracující s protokolem HTTP), je v tomto případě výsledek vyjádřen například ve formátu XML (Beckett, Broekstra, 2008).

Do dotazu můžeme přidat další omezení pomocí konstruktu `FILTER`, kde můžeme například porovnávat literály (včetně regulárních výrazů), provádět logické a matematické operace, testovat vlastnosti proměnných apod. V dotazu též můžeme specifikovat „volitelné" trojice pomocí klíčového slova `OPTIONAL`. Pokud jsou takovéto trojice nale-

zeny, je provedeno navázání proměnných; pokud nejsou nalezeny, jsou přesto navázány proměnné ze zbytku dotazu. Například v uvedeném dotazu můžeme říci, že domovská stránka nás bude zajímat jen v případě, že pro daného člověka existuje, ale i když ji v našich datech nenalezneme, chceme přesto e-mail tohoto člověka. Dalším klíčovým slovem je `UNION`. Pomocí něho můžeme vyjádřit podmínku *nebo* – ve výsledku pak nalezneme sjednocení hodnot vyhovujících jednotlivým vzorům. Podobně jako v SQL můžeme výsledky řadit (`ORDER BY`), chtít jen jedinečné výsledky (`DISTINCT`), popř. jen část výsledků (`OFFSET` a `LIMIT`). Mimo dotaz `SELECT` lze použít i dotaz `ASK` – výsledkem je pak odpověď `yes`, pokud byl nalezen hledaný vzor, v opačném případě je odpověď `no`.

SPARQL ale není jen čistě dotazovací jazyk. Umožňuje též konstruovat nové grafy ve formátu RDF. Příkladem je dotaz vytvářející graf, který vyjadřuje to, že Alena zná všechny ve věku 30 let:

```
CONSTRUCT { data:Alena foaf:knows ?person. }
WHERE { [ a foaf:Person; foaf:age ?age ]. FILTER (?age = 30). }
```

Druhá část je shodná s dotazem typu `SELECT`, v první části ale říkáme, jaký graf se má zkonstruovat (tato část může být složitější než jen jedna trojice). Výsledkem dotazu je pak samozřejmě graf ve formátu RDF. Tento typ dotazů může být velmi užitečný pro integraci dat, dají se s ním konstruovat popisy dat v podobě potřebné pro systém, který používá jiný slovník.

Posledním typem dotazu je `DESCRIBE`, kde se ptáme na jeden zdroj (resource) a jako výsledek očekáváme jeho popis, například ve formě části grafu RDF, který daný zdroj obklopuje.

Dosud uvedené příklady ilustrovaly dotazování nad daty v jazyce RDF. Vzhledem k tomu, že i jazyky RDFS a OWL jsou ukládány v tomto jazyce, můžeme jazyk SPARQL použít i pro dotazování v ontologiích a na nich založených bází dat v jazyce OWL. Příkladem dotazu na podtřídy třídy `VegetariánskáPizza` je:

```
SELECT ?pizza WHERE { ?pizza rdfs:subClassOf
    pizza:VegetariánskáPizza}
```

Tento přístup s sebou ale nese dva problémy. První je oblast syntaxe – jak již jsme viděli, syntax zápisu OWL v RDF může být velmi složitá, což s sebou přináší komplikované dotazy. V aplikacích jsou ale dotazy generovány, nepíše je tedy člověk. Druhým problémem je otázka automatického uvažování. V našem příkladě se určitě bude lišit odpověď dotazu na grafu bez provedeného uvažování a na grafu se spočítanými důsledky. Oba tyto problémy se snaží řešit jazyk SPARQL DL (Sirin, Parsia, 2007) – upravuje totiž syntax jazyka SPARQL s možností jednodušší formulace dotazů pro jazyk OWL a umožňuje předkládat dotaz tak, aby se na jeho zodpovězení přímo podílel i odvozovací stroj.

2.2.6 Spojení s dokumenty na webu

Dosud jsme se příliš nezabývali tím, jak uvedené technologie dostat do webových dokumentů tak, aby bylo možné vytvářet aplikace uvedené v úvodu kapitoly. Základem je anotování dokumentů ve formátu HTML, tedy popis částí takovýchto dokumentů tak,

aby obsah byl srozumitelný pro systémy, které jsou schopny pracovat s jazyky RDF a popřípadě OWL.

Pro vkládání strukturovaných informací lze používat takzvané mikroformáty, které využívají strukturu dokumentu v HTML tak, že dodatečné sémantické informace jsou vloženy přímo do elementů jazyka HTML. Pro mikroformáty existují slovníky[5], které umožňují zpracovávání informací z různých zdrojů. Podobný přístup se objevuje i ve specifikaci jazyka HTML 5, kde je přidáno několik nových atributů pro vkládání mikrodat. Pro vkládání anotací přímo v podobě RDF existuje doporučení RDFa (Adida a kol., 2008), které rozšiřuje XHTML o atributy pro vkládání anotací přímo do elementů dokumentu HTML. Základní rozdíl mezi mikroformáty a RDFa je v možnosti rozšiřovat a kombinovat dokumenty. Zatímco mikroformáty definují jak syntax, tak sémantiku (přiřazují význam konkrétním atributům), RDFa pouze říká, jak vložit informaci v RDF. Popis vlastní sémantiky může být (a často je) uložen v odděleném dokumentu, například ve formátu OWL. U RDFa lze tedy definovat a použít jakoukoliv ontologii bez hrozby možných záměn identifikátorů, jako je tomu u mikroformátů. RDFa je tedy postup vhodnější pro vnoření do architektury sémantického webu. Pro extrakci metadat v podobě mikroformátů (a jiných podobných způsobů) do dat v RDF je standardizován mechanismus GRDDL (Connolly, 2007). Vstupem tohoto mechanismu je anotovaný dokument v XML spolu s popisem transformace, přičemž tyto popisy jsou pro nejpoužívanější mikroformáty již připraveny. Výstupem je pak dokument s metadaty ve formátu RDF.

V aplikacích lze samozřejmě anotovat dokumenty manuálně. V tomto případě člověk, který zná nějaký slovník (popř. používá nástroj pro anotování, který slovník v nějaké podobě nabízí), manuálně přidává anotace k psanému textu. Tento postup je sice možný, ale těžko použitelný ve velkém měřítku. Pro automatické anotování lze použít v zásadě dva přístupy: generovat popisy z již existujících struktur dat nebo extrahovat anotace z existujícího textu. Vzhledem k tomu, že velká část webu je generovaná z databází, je určitě možné do generovaných dokumentů ve formátu HTML dodat i anotace v uvedených formách. Tento přístup má výhodu ve větší škálovatelnosti. Struktura dat je často známa i třeba jen z databáze, takže dodání takovýchto informací nemusí být nijak pracné. I když se může zdát, že tímto způsobem nebude dodaná informace příliš podrobná, je třeba si uvědomit, že v původním webovém dokumentu není taková použitelná informace žádná. I jen zdánlivě triviální anotace může přinést ohromný rozdíl v možnostech vyhledávání a kombinování dokumentů. Druhým uvedeným postupem, tedy extrakcí anotací z existujících dokumentů a automatickým anotováním, se podrobně zabývá kapitola 6.

2.3 Zpracování znalostí na sémantickém webu

Podívejme se nyní na zpracování znalostí a informací reprezentovaných v podobě uvedené v předchozím textu, kde jsme již o využití a zpracování znalostí do určité míry diskutovali. Podívejme se nyní na oblasti, které se dají oddělit od popisu architektury sémantického webu. Nejprve stručně popíšeme editory a prohlížeče ontologií, poté uvedeme ilustraci automatického uvažování. Protože na sémantickém webu se už z principu

[5] Viz http://microformats.org/ – například vizitky, kalendáře, licence apod.

předpokládá použití mnoha různých ontologií, popíšeme též detailněji problematiku jejich kombinování a mapování.

2.3.1 Editory ontologií v jazyce OWL

Pro ruční vytváření nebo úpravy ontologií v jazyce OWL jsou k dispozici uživatelsky přívětivé editory. Ty mimo vlastní editování ontologie umožňují zejména používat automatické uvažování. To je důležité pro kontrolu konzistence, ale také pro odvození a kontrolu faktů, které z ontologie vyplývají. Často je možné vidět i to, jak se k danému faktu dospělo – to je důležité nejen pro odstraňování chyb v ontologiích. Dále některé nástroje poskytují různé možnosti vizualizace.

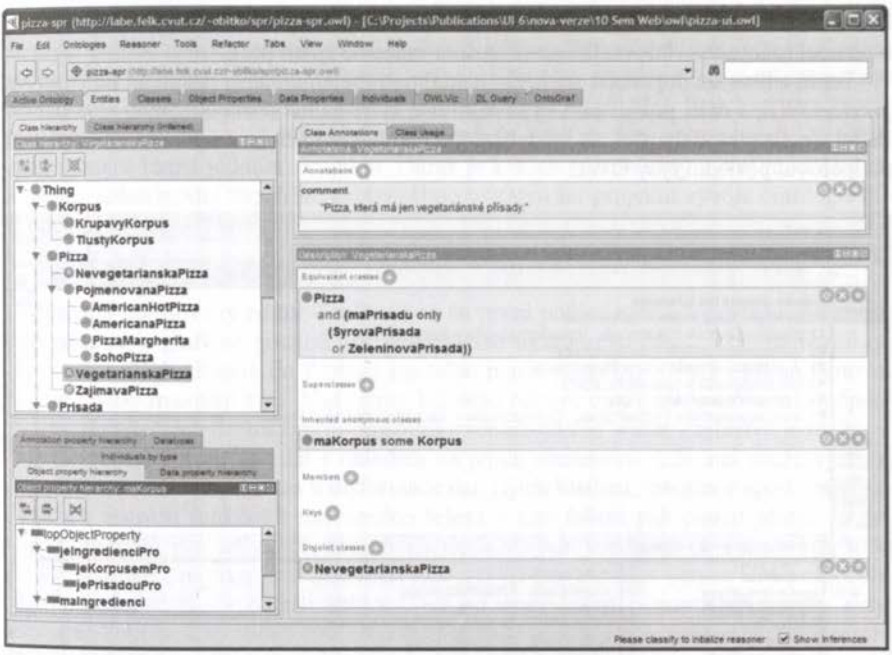
Asi nejznámějšími editory jsou Protégé[6], Swoop[7], NeOn Toolkit[8] a TopBraid[9]. První tři jsou dostupné zdarma, včetně zdrojových textů, pro poslední je zdarma pouze základní edice.

2.3.1.1 Protégé

Zřejmě nejrozšířenější a nejpopulárnější je dnes editor Protégé vyvíjený na Stanfordově univerzitě. Původní editor pro ontologie a znalostní báze založené na rámcích byl postupně přizpůsoben vývoji OWL. Jeho architektura umožňuje přidávat různá rozšíření (pluginy[10]), například pro vizualizaci, uvažování, správu větších ontologických projektů, zpracování přirozeného jazyka, hledání, editování pravidel nebo například popisů sémantických webových služeb atd. Pro seznámení s editorem Protégé lze doporučit tutoriál (Horridge, 2009). Ukázku prostředí představujeme na obr. 2.8, kde vidíme editování ontologie popisující pizzy, konkrétně popis třídy `VegetariánskáPizza`. Základem editoru jsou jeho první záložky znázorněné na obrázku – popis ontologie včetně importovaných ontologií, souhrnný pohled na různé entity a dále záložky specializované na třídy, objektové vlastnosti, datové vlastnosti a individua. Další záložky obsahují rozšíření týkající se v tomto případě vizualizace a dotazování.

Jak je vidět z obr. 2.8, editor v pravé části ukazuje hierarchii entit (tříd, objektových vlastností, datových vlastností) – taxonomie entit je obvykle jádrem ontologie. Tato hierarchie může zobrazovat buď jen to, co je v editovaném dokumentu uvedeno explicitně, například pomocí vztahu `rdfs:subClassOf`, nebo lze pomocí strojového uvažování zobrazit hierarchii dle klasifikace založené na celém popisu entit. V pravé části lze pak upravovat detaily pro vybranou entitu. Například pro třídu máme možnost upravit anotaci a její popis v podobě již zmíněné v této kapitole, tedy mimo jiné vybrat ekvivalentní nebo disjunktní třídy, přidat omezení ve formě omezení použití vlastností, zařadit do taxonomie individua, která do třídy patří, nebo z ní dotyčná individua vybrat. Všechny takovéto axiomy lze zvlášť anotovat. Dále zřetelně vidíme, co z definice je pro třídu specifické, co je zděděné a co je přidané pomocí strojového odvozování.

[6] http://protege.stanford.edu/
[7] http://code.google.com/p/swoop/
[8] http://neon-toolkit.org/wiki/Main_Page
[9] http://www.topquadrant.com/products/TB_Composer.html
[10] http://protegewiki.stanford.edu/wiki/Protege_Plugin_Library



Obr. 2.8 Editor Protégé ve verzi 4.1 s příkladem definice třídy `VegetariánskáPizza`.

Vztahy mezi entitami lze prohlížet i v grafické podobě. Výraznou pomocí jsou různé nástroje pro často se opakující vzory při vytváření ontologie – například přidání tříd, které jsou navzájem disjunktní. Tento editor je ve své základní podobě (bez podpory různých doplňkových modulů) dostupný i ve webovém prostředí[11], ontologie lze tak tedy ve známém uživatelském rozhraní editovat i přímo ve webovém prohlížeči.

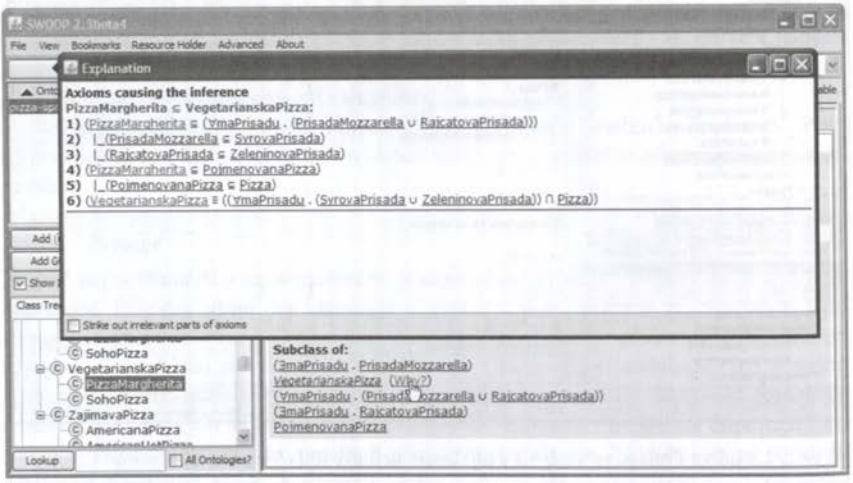
2.3.1.2 Ostatní editory

Editor SWOOP má na rozdíl od editoru Protégé uživatelské rozhraní, které připomíná prostředí webu. V podstatě vše editované ontologii je zároveň odkazem a můžeme se tak rychle dostat k jiné entitě. Tento editor také umožňuje prohlížet ontologie v různých syntaxích jazyka OWL, což může být někdy výhodné – Protégé uživatele od zápisu ontologie zcela odstiňuje, i když samozřejmě import a export v různých formátech podporuje, ale pokud je v zápise ontologie nějaký problém, najdeme ho spíše pomocí editoru SWOOP. Editor SWOOP je také mnohem těsněji integrován s uvažovacím systémem Pellet[12] (Sirin a kol., 2007) a jeho zajímavým rysem je využití tohoto systému pro vysvětlování inference. Na obr. 2.9 je například vidět, jak editor vysvětluje to, že třída `PizzaMargherita` je podtřídou třídy `VegetariánskáPizza` prostřednictvím posloupnosti axiomů, které k tomuto závěru vedly. Toto je velmi užitečné pro hledání ne-

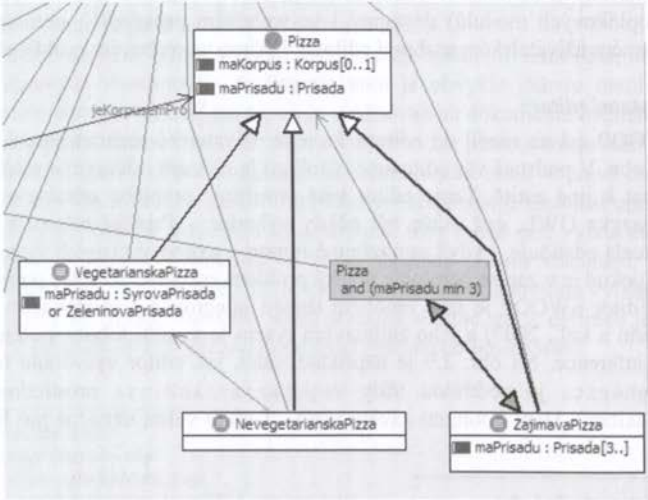
[11] http://webprotege.stanford.edu/
[12] http://clarkparsia.com/pellet/

konzistencí – nekonzistentní třída se nám zařadí jako podtřída třídy `owl:Nothing` a pomocí tohoto vysvětlovacího mechanismu můžeme zjistit proč.

Tento editor lze doporučit zejména, pokud potřebujeme menší odstínění od vnitřností jazyka OWL, a dále, pokud nám vyhovují rysy, které nabízí těsná integrace se systémem Pellet. Je ale také třeba říci, že již delší dobu nebyl tento editor aktualizován, nemusí tak podporovat poslední rysy OWL.



Obr. 2.9 Editor SWOOP vysvětluje, proč je třída `PizzaMargherita` podtřídou třídy `VegetariánskáPizza`.



Obr. 2.10 Grafické znázornění vazeb v editoru TopBraid.

Dalším editorem je NeOnToolkit, který byl vyvinutý v rámci evropského projektu. Na první pohled nabízí rozhraní velmi podobné editoru Protégé, ale jeho síla je v orientaci na větší ontologie, včetně možnosti práce s několika provázanými ontologiemi najednou, a na odpovídající výkon, včetně automatického uvažování. Podobně jako Protégé nabízí spoustu přídavných modulů, tyto moduly jsou ale lépe integrované – je vidět jejich vývoj v rámci jednoho projektu. Důraz je kladen na společný vývoj modulárních ontologií, jeden modul například nabízí přímo rozfázování projektu vývoje ontologie dle předpokládaných omezení a počátečních znalostí. I když je v současnosti editor Protégé populárnější, zejména pro vývoj větších ontologií lze určitě editor NeOnToolkit doporučit.

Poslední popisovaný editor TopBraid se na první pohled vyznačuje především možností editovat v grafické podobě, jak je například ukázáno na obr. 2.10 – na rozdíl od běžně dostupných pluginů do Protégé lze takto nejen vizualizovat, ale i přímo editovat ontologii nebo znalostní bázi. Nad rámec běžného editoru ontologií je nabízena podpora přístupu do relačních databází přes prostředky sémantického webu. Zajímavá je možnost vizuálně popsat zpracování dat s ohledem na jejich sémantiku. Uživatel může vytvořit diagram, ve kterém je popsána transformace dat, jejich hledání, integrace apod., přičemž lze přímo testovat funkčnost navrženého řešení a toto řešení pak použít přímo ve své aplikaci. Kromě pro uživatele popsaných dodatečných vlastností je editor TopBraid vhodný zejména pro programátory, kteří jsou zvyklí modelovat v jazyce UML a podobný přístup vyžadují i v oblasti aplikací, jež se nějak týkají sémantického webu nebo ontologií. TopBraid je cílený pro komerční použití a je k němu nabízena podpora pro vývoj celých částí aplikací, nejedná se tedy jen o prostý editor.

2.3.2 Strojové odvozování

Jak jsme již zmínili dříve, pro využití ontologií je klíčové, že mohou být zpracovávány automaticky. O ukázkách odvozování nových znalostí jsme se již zmínili, ilustrujme nyní na několika příkladech možná využití.

Pro jazyky OWL a RDFS jsou k dispozici nástroje pro strojové odvozování, tedy nástroje, které umožňují odvozovat logické důsledky z množiny axiomů. Tato množina může zahrnovat jen ontologii nebo znalostní bázi (instance) spolu s ontologií. Jak jsme již viděli, editory OWL mají často takovýto nějaký nástroj integrován, což je užitečné pro správu ontologie. Klasickým problémem při vytváření ontologie je nezamýšlené vytvoření třídy, která je nekonzistentní, čili nemůže obsahovat žádné instance. Toto je typickým důsledkem nepromyšlení všech důsledků – například definování vegetariánské pizzy jako pizzy, která obsahuje sýrové přísady, a poté přidání sýrové přísady, která ale obsahuje i maso. Takovéto nezamýšlené důsledky by měl editor rychle rozpoznat. V práci (Horrocks, 2008) autor uvádí příklady nalezených podobných problémů v ontologiích popisujících geografické objekty nebo v ontologiích z medicínské oblasti. Pro odstranění závad typu nekonzistence nebo nezamýšlené klasifikace třídy do určité oblasti ontologie je užitečné mít i možnost vysvětlení. Příklad takovéhoto vysvětlení byl již ilustrován na obr. 2.10.

V případě aplikací lze uvažování použít obdobným způsobem. Například při dotazování na instance třídy pizza chceme jako odpověď nejen explicitně označené instance této třídy, ale i všechna individua, která mají korpus pizzy a přísady. Tak jako v případě editorů je velmi důležitá možnost vysvětlování odvozených tvrzení. Vysvětlení je po-

skytnuto typicky ve formě minimální (nebo alespoň dostatečně malé) množiny axiomů, ze kterých lze dané tvrzení odvodit. Pokud je aplikační doména popsaná čistě ontologií, popř. pravidly v nějakém pravidlovém jazyce, znamenají změny v aplikaci často jen úpravu ontologie a pravidel – více viz příklad aplikace v kapitole 5.

Příklady uvedené v tomto oddílu byly zamýšleny jako ilustrativní, pro formální popis včetně konkrétních algoritmů odkazujeme čtenáře na kapitolu 3. Podotkněme, že ačkoliv v současné době existuje vícero nástrojů pro strojové odvozování (viz kapitola 7), které efektivně umožňují uvedená použití, zůstává uvažování v rozsáhlejších ontologiích a v rámci heterogenního a nekonzistentního webu stále předmětem výzkumu.

2.3.3 Vyhledávání a mapování ontologií

Se vzrůstajícím počtem znalostních ontologií přítomných na sémantickém webu roste také potřeba jejich vyhledávání a také zpřístupňování různým skupinám uživatelů, což zahrnuje vzájemné kombinování či mapování ontologií. V této části kapitoly přiblížíme existující postupy pro vyhledávání ontologií a pro jejich mapování, včetně příkladů konkrétních nástrojů a aplikací.

2.3.3.1 Metody a nástroje pro vyhledávání ontologií

Popišme nejprve vybrané nástroje pro vyhledávání ontologií a jiných sémantických dokumentů[13].

S představováním začneme u nejmladšího nástroje pro vyhledávání ontologií, zvaného *Watson*, viz (http://watson.kmi.open.ac.uk/), (d'Aquin a kol., 2007). Ze tří uvedených vyhledávačů se zdá mít nejbouřlivější vývoj v podobě obohacování o nové funkce. Watson bývá často využíván různými sémantickými aplikacemi, jimiž jsou například *Scarlet* (nástroj pro zjišťování sémantických relací mezi koncepty pomocí různých online ontologií), *Cupboard* (systém pro online kolektivní správu ontologií, autoři obou nástrojů jsou ze stejného institutu jako autoři nástroje Watson). Watson nabízí sémantické vyhledávání ontologií, které předtím na internetu sesbíral (crawling) a zaindexoval (indexing). Zaindexováno je okolo 500 tisíc sémantických dokumentů. Uživatelé mohou ontologie vyhledávat přes webové rozhraní zadáním klíčových slov z ontologie (jméno třídy, jméno vlastnosti, jméno instance, komentář a další možnosti). Výsledkem vyhledávání je množina odkazů na sémantické dokumenty setříděných podle metrik odrážejících kvalitu sémantického obsahu. Důležitou možností je zobrazení obsahu ontologie. Novějšími funkcemi je možnost dotazování pomocí jazyka SPARQL a extrakce klíčových konceptů ontologie (Peroni, Motta, d'Aquin, 2008). Vedle webového rozhraní nabízí Watson i programátorské rozhraní v jazyce Java, které umožňuje pracovat se všemi zaindexovanými ontologiemi. Nástroj Watson je také možné jako plugin přidat do oblíbeného ontologického editoru Protégé či editoru NeOnToolkit pro podporu tvůrců ontologií při znovupoužití ontologických prvků z jiných ontologií.

OntoSelect (http://olp.dfki.de/ontoselect/), (Buitelaar, 2004) nabízí podobné základní možnosti vyhledávání ontologií. Kromě zadání klíčového slova pro vyhledání ontologií na webu je možné také vyhledávat podle tématu. V tom případě se dotaz uživatele pro

[13] Sémantickými dokumenty chápeme takové dokumenty, které jsou zapsány v RDF či v jazyce OWL.

vyhledání ontologie sestaví podle klíčových slov vybraných ze stránky Wikipedie pojednávající o daném tématu. V současné době je v knihovně OntoSelect 1530 ontologií.

Swoogle (http://swoogle.umbc.edu/), (Li Ding, 2004) rovněž umožňuje vyhledávat ontologie na webu. V současné době je možné prohledávat údajně přes 10 tisíc ontologií. Kromě ontologií je možné hledat i jiné dokumenty, které obsahují data ve formátu RDF. Takových dokumentů nyní Swoogle indexuje 1,4 milionu, což odpovídá více než jedné miliardě trojic RDF. Je možné vyhledávat pomocí klíčových slov sémantické webové dokumenty i jednotlivé ontologické prvky. Výsledky jsou obohaceny o metadata a je možné okamžitě zkontrolovat konzistenci pomocí reasoneru Pellet.

Poslední nástroj, o kterém bychom se na tomto místě ještě zmínili, je sémantický vyhledávač *Sindice* (http://sindice.com/). Tento nástroj není primárně určen pro vyhledávání ontologií, nýbrž pro vyhledávání dat RDF na webových stránkách. Je možné se dotazovat na jednotlivé termíny či na celá URI. Vizualizaci nalezených odpovědí může například vytvářet prohlížeč sig.ma (http://sig.ma/). K dispozici je též „rozšířený dotazovací jazyk", který umožňuje dotazování na trojice RDF. Vyhledávač Sindice je využíván jako důležitý stavební kámen sémantických aplikací, které mohou nástroj integrovat pomocí existujícího API.

2.3.3.2 Mapování a integrace ontologií

Ontologie tematicky spadající do jedné domény mohou vykazovat různou míru odlišnosti (heterogenity), což přirozeně pramení z různých způsobů modelování (viz kapitola 1). Zachycení určitého úseku reality může být návrh od návrhu rozdílné, aniž by existoval jediný správný způsob. Faktory, které významně přispívají k heterogenitě, se dotýkají například účelu návrhu ontologie, perspektivy zaujaté navrhovatelem ontologie, dále jeho odborných znalostí, a dokonce i osobních názorů. Vedle subjektivních vlivů zde hrají svou roli i objektivní vlivy, jimiž mohou například být použitá metodologie pro návrh a editor pro implementaci ontologií, dodržované jmenné konvence platné v rámci dané komunity a aplikované návrhové vzory.

Heterogenní charakter ontologií vede k vzájemně odděleným, explicitně nepropojeným ontologiím nebo jejich částí. Oblast, která se tímto propojováním zabývá, se nazývá *mapování ontologií*. Českým označením mapování ontologií (Euzanat, Shvaiko, 2007) rozumíme jednak proces (matching) zjišťování vztahů mezi prvky ontologií a jednak samotný výsledek tohoto procesu (alignment). S ohledem na množství heterogenních ontologií je žádoucí dosáhnout jisté míry automatizace. Základním vstupem do procesu jsou (alespoň) dvě ontologie; výstupem pak jejich množina mapování (alignment). Množina mapování se skládá z jednotlivých dílčích mapování (correspondence, korespondence) mezi dvěma prvky ontologií. Těmito prvky mohou být instance, třídy, vlastnosti a také celé komplexní formule. Hledané vztahy mezi prvky ontologií mohou být skoro jakékoliv. Nejčastějšími bývají vztahy ekvivalence, subsumpce, překrývání (overlapping) a disjunktnosti (disjointness).

Z aplikačního pohledu je zřejmé, že vedle propojování prvků ontologického obsahu je užitečné propojovat jednotlivá data neboli instance. Činnost *mapování instancí* představuje snahu o rozhodnutí, zda dvě instance (jejich popisy) odkazují k témuž objektu, k čemuž se hodí využít informaci zachycenou v ontologickém popisu dat.

2.3.3.3 Aplikace mapování ontologií

Výsledky mapování ontologií nacházejí obecně uplatnění všude tam, kde je potřeba pracovat s různými ontologiemi modelujícími stejnou věcnou oblast. Mezi tyto aplikace patří například *překlad dotazů* (query reformulation), který nastává v případě, že dotaz (query) je specifikován podle jedné určité ontologie, ale má sloužit k dotazování nad daty, která jsou podle jiné určité ontologie. K této situaci dochází například v peer-to-peer sítích, kde je velká míra autonomie jednotlivých uzlů sítě (peers). Další aplikací je *datová migrace* (data migration), při které se provádí převod dat z jednoho systému do systému jiného. V těchto dvou systémech jsou data reprezentována podle různých ontologií. V databázovém prostředí se této aplikaci říká *datová integrace*. Jinou aplikací je *spojování ontologií* (ontology merging), která vede buďto k tomu, že se dvě ontologie spojí do jedné sjednocené ontologie, nebo že je jedna ontologie obohacena o prvky ontologie druhé. Tato aplikace nachází uplatnění například při integrování produktových katalogů či během tzv. evoluce ontologií. V databázovém prostředí se tato aplikace rovná integraci schémat potřebné například při fúzi firem. Mezi další aplikace patří multi-agentní komunikace (viz kapitola 5), hledání a skládání webových služeb a navigace na sémantickém webu.

2.3.3.4 Metody mapování

Metody mapování ontologií hojně čerpají ze starší disciplíny *mapování schémat* (Rahm a Bernstein, 2001), které bylo a stále je důležitým problémem jak v databázovém světě, tak později i světě schémat XML. Metody mapování ontologií můžeme rozdělit podle toho, na jaký typ ontologického obsahu se zaměřují. Dostáváme tak metody, které pracují jen se schématy ontologií, tzn. Tboxem, nebo jen s instancemi, tzn. Aboxem. Nejčastějším případem při mapování je využívání obou součástí ontologie. Druhé často uváděné rozdělení spočívá v zohlednění různých hledisek ontologií (řetězcové, grafové a sémantické). *Řetězcové* metody se zaměřují na části ontologie, ve kterých figurují řetězce jako posloupnost znaků. Řetězce se objevují v popiscích prvků (rdfs:label) a v komentářích, které slouží pro lidského uživatele ontologie. Dále tyto metody často pracují s celým identifikátorem (URI) prvku, či s jeho částí, který slouží pro strojové zpracování. Tyto identifikátory zobrazují například různé editory a prohlížeče ontologií (*Protégé*, *Swoop*, *OWLSight*, *TopBraid*). Na základě řetězcových metod se nejčastěji určuje vztah ekvivalence mezi prvky podle podobnosti daných řetězců. Podobnost, resp. vzdálenost řetězců se určuje pomocí různých metrik – *CharJaccard*, *Jaccard*, *Jaro* – či se využívá editační vzdálenosti slov – *Levenshtein*, *Smith-Waterman* (Cohen a kol., 2003). Při zpracování rozsáhlejších popisků je dále možné využít přístupy *zpracování přirozeného jazyka* (NLP) – tokenizaci, lematizaci a morfologickou analýzu. V rámci zpracování řetězcového aspektu ontologie se také setkáváme s využíváním různých externích lingvistických zdrojů jako například specificky doménově orientovaných tezaurů či obecného WordNetu.

Strukturní metody berou v úvahu strukturu ontologií. Nejčastějšími strukturními metodami jsou metody grafové, které s ontologiemi pracují jako s orientovanými cyklickými grafy, kde jednotlivé koncepty odpovídají uzlům a vlastnosti odpovídají hranám. Tyto metody často staví propagování podobnosti na různých pravidlech týkajících se struktury. Například dva prvky jsou ve vztahu ekvivalence, pokud jsou v tomto vztahu jejich sousedé.

Třetím aspektem, který se při mapování ontologií využívá, je hledisko sémantické. *Sémantické* metody jsou založeny na práci s ontologiemi a jejich prvky jako s logickými formulemi deskripční logiky. Uvedené metody mohou být využity i pro mapování instancí.

V různých systémech mapování ontologií bývá implementováno více různých typů metod, které se vzájemně doplňují. V následujícím odstavci uvádíme příklad dvou systémů mapování ontologií a jeden systém mapování instancí, které dosahují velmi dobrých výsledků.

2.3.3.5 Příklady systémů

Falcon-AO (Hu, Qu, 2007) obsahuje tři elementární komponenty mapování, které jsou vzájemně zkombinovány. Komponenta *V-Doc* určuje podobnost prvků prostřednictvím porovnání vektorů reprezentujících jednotlivé prvky. Tyto vektory jsou vytvořeny z řetězců názvů prvků a jejich sousedů. Díky této technice se při hledání korespondencí neuvažují prvky izolovaně, ale bere se v úvahu i jejich okolí. Druhá komponenta *I-Sub* je řetězcového typu, která staví nejen na podobnosti řetězců, ale také na jejich rozdílnosti. Poslední komponentou mapování je grafová *GMO*, která zjišťuje strukturální podobnost mezi bipartitními grafy *RDF* reprezentujícími ontologie. Pro rozsáhlé ontologie tento systém nabízí speciální komponentu *PBM* rozdělující rozsáhlé ontologie na menší části, které se pak mezi sebou mapují pomocí tří uvedených komponent mapování.

Systém *ASMOV* (Yves, Kabuka, 2008) mapuje ontologie ve třech fázích, jimiž jsou: předzpracování, iterativní proces a sémantická validace výsledků. Ve fázi předzpracování jsou ontologie načteny do systému, anotovány pomocí lexikálních systémů, jako je například WordNet, a analyzovány, což slouží pro potřeby nastavení parametrů mapování. V iterativní fázi jsou počítány podobnosti mezi prvky ontologií s využitím čtyř hledisek: textových popisků prvků (slovního označení prvku, komentáře atd.), externí struktury (rodiče a děti), interní struktury (vlastnosti) a podobnosti individuí (instancí). Na konci každé iterace probíhá validační proces, kde se kontroluje konzistence nalezených korespondencí. Nekonzistentní korespondence se z mapování odstraní. Systém také umí pracovat s počátečním částečným mapováním jako vstupem do celého procesu mapování.

Silk (Volz a kol., 2009) je systém pro mapování instancí. Mapování probíhá semiautomaticky prostřednictvím silného „specifikačního" jazyka (Silk Link Specification Language). Uživatel může specifikovat typ hledaného vztahu mezi instancemi (např. owl:sameAs) a podmínku, za které vztah bude platit. Vybrané klasické mapovací algoritmy je možné kombinovat, např. vážený průměr numerických a řetězcových porovnání. Proces mapování je možné urychlit přidáním omezení, jež vybírá páry instancí, které podstoupí důkladnější způsoby porovnání. Přístup k datům je realizován prostřednictvím „SPARQL endpointů"

2.3.3.6 Vyhodnocování výsledků mapování ontologií

Důležitou součástí životního cyklu výsledků mapování ontologií je vyhodnocení jejich kvality. Zjišťování kvality výsledků různých systémů a jejich porovnání je jedním z cílů každoročně pořádané iniciativy *Ontology Alignment Evaluation Initiative*[14] (OAEI).

[14] http://oaei.ontologymatching.org/

Korespondence nalezené systémem lze porovnat s referenčním mapováním (gold standard) a vyvodit z toho tradiční míry – přesnost a úplnost, které jsou převzaty z oblasti *Information Retrieval*. Výhodou použití těchto měr je jejich snadné zjištění. Na druhou stranu však tyto míry neberou v úvahu ani sémantiku korespondencí, ani sémantiku ontologií, tzn., že se porovnává (ne)přítomnost korespondencí mapování od systému v referenčním mapování a neberou se v úvahu různé důsledky korespondencí či ontologických popisů prvků v ontologiích. V práci (David, Euzenat, 2008) se zmiňované nedostatky napravují zavedením *sémantické přesnosti* a *úplnosti*, které jsou však stále závislé na syntaktické formě mapování, proto se také dále definuje podoba mapování v normální formě, čímž se řeší problémy plynoucí z různé syntaktické podoby mapování.

Jelikož však je referenční mapování k dispozici jen zřídka (jeho tvorba je náročný a proto nákladný proces), nelze si pro určování kvality ontologií vždy vystačit s tímto porovnáním. Proto se využívá náhodných výběrů, které se ručně anotují, a spočítaná přesnost a úplnost slouží jako odhad kvality celé množiny mapování. Určí-li se pravděpodobnostní rozdělení korespondencí, můžeme spočítat chybové odchylky metrik na výběru s ohledem na celek (van Hage a kol., 2007).

Vedle tradičních metrik můžeme k posouzení kvality mapování ontologií použít například metriky, které jsou postaveny na míře nekonzistence způsobené aplikováním mapování pro spojení ontologií. V práci (Meilicke, Stuckenschmidt, 2008) jsou definovány metriky založené jednak na porovnání nesplnitelných konceptů v ontologiích před mapováním a po něm a jednak na porovnání počtu korespondencí, které se musí smazat, aby bylo opět dosaženo konzistentního stavu a počtu korespondencí v původním mapování.

2.3.3.7 Současné trendy v mapování ontologií

Mapování ontologií je oblast, které je v současné době věnována velká pozornost. Pro úspěšné řešení problému mapování ontologií se využívá poznatků z různých oblastí – databáze, induktivního logického programování, logiky, kognitivních aspektů, umělé inteligence aj.

V současné době se výzkum v oblasti mapování ontologií (Shvaiko, Euzenat, 2008) zaměřuje na mapování rozsáhlých ontologií (large scale), což se odráží také v jedné sekci OAEI. Škálovatelnost je obecně důležitým hlediskem všech technologií sémantického webu. Úzce souvisí s výkonností (performance) nástrojů. Optimalizaci fungování systémů mapování ontologií se zatím věnuje jen několik málo systémů (Falcon-AO, ASMOV, RIMOM). Důvodem je nejspíše fakt, že primárním cílem tvůrců systémů je dosažená kvalita ontologií. Ta je také předmětem vyhodnocení v rámci OAEI; čas zpracování se zatím neporovnává (s výjimkou sekce anatomy). Současným rysem přístupů je také důraz kladený na využívání externích (znalostí) zdrojů a na způsoby jejich automatického vyhledání. Důležitou stále otevřenou otázkou zůstává jednotný přístup k míře jistoty mapování, která může být různého druhu (pravděpodobnostní, fuzzy apod.). Zájem různých vědeckých pracovišť přitahuje problematika kombinování různých nástrojů mapování, popřípadě výběr jednoho nejvhodnějšího nástroje. Objevují se také experimenty využívání tranzitivnosti vztahů korespondencí mezi entitami. V poslední době se také začíná pozornost věnovat uživateli, jehož vstup je často velmi důležitou nutnou podmínkou pro kvalitní výsledky mapování ontologií. Vznikají komunitní systémy umožňující spravovat mapování a diskutovat o jejich platnosti. Pozornost se nově také

upírá na možnosti získat mapování ontologií za využití webu dat jako výsledku iniciativy „Linked data". Vedle toho je také snahou zapojit a propojit metody mapování ontologií v této iniciativě.

2.4 Praktické aplikace sémantického webu

V tomto oddílu si popíšeme hlavní současné „záchytné body" sémantického webu v praktických webových aplikacích využívaných velkým počtem uživatelů, včetně příslušných (zpravidla jednoduchých) ontologií, resp. slovníků.

Poznamenejme, že jako součást sémantického webu můžeme na druhé straně chápat i aplikace omezené na menší okruh zpravidla odborněji zaměřených uživatelů, kteří využívají i ontologie rozsáhlé – zejména jde o oblast biomedicíny. Těmi se zde podrobněji zabývat nebudeme, čtenáře můžeme odkázat například na Bioportal (http://bioportal.bioontology.org/).

2.4.1 Sociální sítě

Když se podíváme na počty dokumentů sémantického webu dostupné na internetu, největší zastoupení mají z ontologického hlediska nepříliš zajímavá metadata podle RSS verze 1.0 (http://web.resource.org/rss/1.0/spec) zachycující zejména titulek, popis a položky webového informačního kanálu. Hned za nimi však následují *osobní profily* vytvořené podle ontologického slovníku *FOAF* („Friend-of-a-Friend", http://www.foaf-project.org/), se kterým jsme se již několikrát setkali v příkladech. Mohli jsme si přitom všimnout ústředního predikátu tohoto slovníku, foaf:knows, který umožňuje propojovat profily navzájem a vytvářet z nich otevřenou sociální síť. Řada aplikací proto FOAF využívá mimo jiné pro analýzy vztahů mezi odbornými komunitami.

Zatímco FOAF umožňuje popisovat lidi jako takové, na něj navázaná ontologie *SIOC* (Semantically-Interlinked Online Communities, http://sioc-project.org/) se zaměřuje na zapojení lidí do elektronických diskusí, protože zachycuje vztahy mezi různými webovými fóry a diskusními skupinami, jejich odběrateli a přispěvateli, jednotlivými příspěvky a tematickými značkami, které jsou příspěvkům přiřazeny. SIOC tak umožňuje diskusním fórům dosáhnout přímočaře interoperabilitu.

Zatímco FOAF a SIOC vznikly z iniciativy samotné komunity sémantického webu, protokol *OpenGraph* (http://ogp.me/) pochází z dílny „velkého hráče" webových sociálních sítí, společnosti FaceBook. Protokol předpokládá anotování (osobních, ale i produktových) webových stránek sadou vlastností reprezentovaných pomocí RDFa a vyjadřujících základní vlastnosti stránky – její titulek, (tematický) typ, URL, popř. i kontaktní a geolokační informace apod. Pokud návštěvník stránky klepne na tlačítko „Like", které je na ní umístěno, stránka se automaticky začlení do sítě FaceBooku.

2.4.2 Hesláře a tezaury

Vedle ontologií definující třídy objektů reálného světa je na webu často třeba pracovat se soubory pojmů, ke kterým přímo instance přiřadit nelze (můžeme říci, že tyto pojmy mají pouze intensionální a nikoliv extensionální sémantiku). Jde typicky o různé hesláře a tezaury (o nich byla řeč v předchozí kapitole) vznikající v knihovnickém prostředí

i mimo ně. Pro tento typ zdrojů může být vhodné použít reprezentaci založenou na hierarchicky uspořádané síti instancí a nikoliv na subsumpční hierarchii tříd. Proto byl navržen slovník *SKOS* (Simple Knowledge Organization System, http://www.w3.org/2004/02/skos/), který staví na tezaurových asociacích typu deskriptor/nedeskriptor (prefLabel, altLabel), obecnější/specifičtější pojem (broader, narrower) nebo související pojem (related). SKOS je specifikací W3C a je používán řadou hierarchických zdrojů; v tuzemsku například *Polytematickým strukturovaným heslářem* Národní technické knihovny v Praze (http://psh.ntkcz.cz/skos/).

2.4.3 Kolaborativní publikování

Vedle tvorby sociálních sítí patří mezi hlavní prvky tzv. Webu 2.0 kolaborativní publikování článků a jiných dokumentů, zejména pomocí technologií wiki a CMS. I v této oblasti se v současnosti ontologie a technologie sémantického webu uplatňují.

Tak jako sémantický web na obecné úrovni přidává explicitní strukturované metainformace k textovým informacím na běžném webu, principem *sémantických wiki* je přidání takových metainformací zachycených ve formě wiki k textovým informacím. Postupně vykrystalizovaly dva hlavní přístupy: textově orientované systémy wiki (jako je *SemanticMediaWiki*, http://semantic-mediawiki.org) umožňují doplňovat sémantické anotace přímo v rámci textů zapsaných syntaxí wiki (pomocí rozšíření této syntaxe), zatímco formulářově orientované systémy wiki (jako je *OntoWiki*, http://ontowiki.net) spíše přenášejí obecné „wiki principy" otevřené editovatelnosti na půdu zpracování strukturovaných informací: uživatel pracuje s formuláři nebo jinými prvky grafického rozhraní, které mají na pozadí reprezentaci RDFa.

Sémantické prvky se postupně prosazují i v rámci druhého rozšířeného prostředku pro tvorbu uživatelského obsahu na WWW – *systémů pro správu obsahu* (CMS). Vlajkovou lodí je mezi nimi Drupal (http://drupal.org/), který ve verzi 7 obsahuje mapování systémových entit (články, uživatele, termíny apod.) na ontologické pojmy z rozšířených slovníků, jako je FOAF, SIOC nebo SKOS.

2.4.4 Vývoj softwaru

Jednu z rozšířených aktivit provozovaných na webu, kolektivní vytváření a údržbu aplikací s otevřeným kódem, podporuje ontologie *DOAP* (Description of a Project, http://trac.usefulinc.com/doap). Zachycuje pojmy, jako jsou domovská stránka projektu, implementační jazyk, osoby podílející se na vývoji, úložiště kódu, licenční podmínky, možnosti stažení, asociované diskusní skupiny apod.

2.4.5 Elektronické obchodování

V současnosti vznikají nejpropracovanější a současně velmi používané sémantické modely pro elektronický obchod v dílně výzkumného týmu prof. Martina Heppa na Univerzitě Bundeswehru v Mnichově (viz http://www.heppnetz.de).

Ontologie *eClassOWL* (http://www.heppnetz.de/projects/eclassowl/) je ontologickou verzí katalogu typů produktů eCl@ss. V kontrastu k malým ontologiím zmíněným v předchozím textu jde o velmi rozměrný model, jenž zachycuje asi 60 tisíc typů produktů a služeb a umožňuje charakterizovat jednotlivé produkty a služby pomocí více než 5 tisíc vlastností. Ontologizace původní taxonomie zahrnovala mimo jiné rozdělení kaž-

dé třídy na dvě (třídu a její podtřídu), z nichž první zahrnuje pouze korektní ontologické instance daného typu entity, zatímco druhá navíc zahrnuje instance, které byly pod původní třídu zahrnuty jen proto, že nějak s korektními instancemi „souvisejí". Například třída „Stroje na zpracování potravin", která je podtřídou třídy „Stroj", se rozdělí na třídu zahrnující pouze stroje a na její nadtřídu, která navíc zahrnuje například služby typu „oprava stroje na zpracování potravin" nebo „náhradní díl stroje na zpracování potravin". Metoda transformace taxonomie je podrobněji popsána v práci (Hepp, Bruijn, 2007).

Zatímco eClassOWL vychází z respektovaného průmyslového standardu, její později vyvinutá alternativa *Product Types Ontology* (http://www.productontology.org/) se opírá o soubor hesel anglické Wikipedie, tedy o využívání výsledku dobrovolné komunitní spolupráce. Každé z asi 300 tisíc hesel je převedeno do podoby třídy jazyka OWL a opatřeno jazykovými nálepkami a textovým popisem (abstrakt z Wikipedie jako hodnota vlastnosti rdfs:comment).

Ontologie *GoodRelations* (viz http://www.heppnetz.de/projects/goodrelations/) je komplementem k eClassOWL či Product Types Ontology (dá se ovšem použít i s libovolnou jinou ontologií typů produktů). Pokrývá problematiku obchodních transakcí, tj. zahrnuje pojmy, jako je cena, způsob platby, způsob dodání nebo záruka. Jejím ústředním konceptem je nabídka („Offering") vztahující se k nějakému produktu, ev. službě („ProductOrService"). Vůči této nabídce může být určitá obchodní entita („BusinessEntity") v pozici nabízejícího (relace „offers") nebo naopak poptávajícího se (relace „seeks"). Nabídka je dále specifikována pomocí ceny („PriceSpecification"). Celkově GoodRelations v současnosti (jaro 2011) zahrnuje asi 30 tříd, 50 objektových vlastností, 40 datových vlastností a 45 individuí. Přestože je GoodRelations poněkud sofistikovanější než zmíněné malé ontologie, a tudíž je objem předběžných znalostí nutných pro její kvalifikované využívání vyšší, míra jejího využívání velmi rychle roste, zřejmě díky prokazatelným finančním benefitům a také dokumentaci, která vyhovuje standardům kvality obvyklým ve firemní praxi. Podle statistik zveřejněných jejími tvůrci (údajně získaných pomocí služby pingthesemanticweb.org) byla tato ontologie již v r. 2009 z hlediska počtu na ni odkazujících dokumentů RDF na druhém místě (600 tisíc) za FOAF (1,2 milionu).

Technologicky nejběžnějším způsobem využití GoodRelations je přímé vložení příslušných informací do firemních webových stránek pomocí jazyka RDFa. Pro tuto aktivitu byly vyvinuty jednak webové nástroje pro ruční vkládání souhrnných informací, jako jsou *GoodRelations Annotator* a *GoodRelations Snippet Generator*, jednak zásuvné moduly do řady populárních softwarových aplikací typu CMS pro elektronické obchody umožňující produkovat RDFa dynamicky.

Za popularitou GoodRelations nesporně stojí také skutečnost, že se k ní vedle tisíců menších firem provozujících internetové obchody otevřeně přihlásila také řada významných komerčních subjektů na straně poskytovatelů dat i provozovatelů vyhledávacích aplikací. Do první kategorie patří americká síť internetových obchodů BestBuy, která s její pomocí vytvořila vlastní šablonu pro popis nabídek a aplikovala ji na sortiment asi 450 tisíc výrobků nabízených v přibližně tisíci obchodech. V oblasti vydavatelské činnosti je průkopníkem používání GoodRelations společnost O'Reilly. Nejznámnější „gigant" v tomto oboru, firma Amazon, se sice zatím pro využívání GoodRelations nevyslovil otevřeně, jeho data přístupná pomocí webového API však jsou snadno

převoditelná do RDF nástrojem URIBurner (http://uriburner.com/) firmy OpenLink. Na straně provozovatelů vyhledávacích aplikací je tradičně podporovatelem GoodRelations zejména Yahoo!, který v rámci programu SearchMonkey začal již v r. 2008 propagovat využívání různých typů strukturovaných dat na webových stránkách, a to vedle mikroformátů právě strukturovaných dat podle standardu RDFa odkazujících na slovníky, jako je GoodRelations. Strukturovaná data slouží v Yahoo! nejen jako prostředek rychlého zobrazení pro koncového uživatele (například přímo v seznamu výsledků vyhledávání můžeme vidět otvírací dobu obchodů a další související informace), ale i jako globální schéma pro konsolidaci obchodních informací od různých poskytovatelů. Nejsilnější hráč v této oblasti, Google, původně navrhl pro oblast obchodních nabídek své vlastní slovníky RDF, v listopadu 2010 však otevřeně ohlásil souběžnou podporu GoodRelations.

Vedle vyhledávacích aplikací pro koncové zákazníky má GoodRelations potenciál i v oblasti obchodování typu B2B. Neklade si ovšem za cíl konkurovat stávajícím jazykům pro elektronickou výměnu dat (EDI), ale spíše je komplementovat v rané fázi výběru obchodních partnerů, kdy mezi subjekty ještě neexistuje přímý kontakt (nebo je natolik nezávazný, že nelze předpokládat výměnu rozsáhlých a částečně i důvěrných informací), popřípadě v situacích kdy firma obchoduje s větším počtem partnerů, ovšem s každým z nich s různou frekvencí.

2.4.6 Linked data

Zatímco zmíněné aplikace mají za svůj základ ontologii (resp. slovník, schéma), na které se sekundárně odvolávají dokumenty vznikající distribuovaně na webu, iniciativa *Linked Data* (http://linkeddata.org/) bere jako výchozí bod rozsáhlé datové soubory a jim přizpůsobuje výběr či návrh nových ontologických zdrojů, na které se položky těchto souborů budou odvolávat. Datové soubory ve formátu RDF vznikají buďto transformací ze strukturovaných databází (přímo nebo vytěžováním webových API), nebo z polostrukturovaných dat (jako jsou tzv. informační boxy ve Wikipedii). Pro potřeby aplikací (i lidských uživatelů) sémantického webu mohou být vystaveny několika způsoby:

- pomocí dotazovací služby (koncového bodu – endpointu) využívajícího jazyk SPARQL;
- jako strukturovaná data zpřístupňovaná pomocí požadavků HTTP s využitím tzv. negociace obsahu: zatímco běžný webový prohlížeč je získá v podobě stránek HTML, softwarová aplikace zaměřená na automatické zpracování dat obdrží pro stejné zdrojové URI přímo trojice ve zvolené syntaxi RDF;
- jako kompletní datový soubor RDF (dump);
- jako data RDFa zanořená do stránek HTML.

Nejvýznamnějším zástupcem zdrojů Linked Data je nepochybně *DBpedia* (http://dbpedia.org), která periodicky vytěžuje informační boxy Wikipedie (které dále doplňuje texty abstraktů článků extrahovanými jako celek). Vzhledem ke svému širokému tematickému pokrytí se často používá jako pojivo pro jiné datové zdroje. DBpedia má svou vlastní rozsáhlou avšak plochou ontologii, která postihuje vlastnosti, kterými jsou entity Wikipedie v informačních boxech popsány.

Vedle všeobecných, „encyklopedických" zdrojů, jako je DBpedia, jsou dalšími významnými kategoriemi Linked Data (popř. Linked Open Data – otevřených zdrojů, které prozatím v rámci Linked Data dominují) zdroje *bibliograficko-publikační* (např. verze

RDF americké Kongresové knihovny, citační databáze Citeseer nebo populární informatické bibliografie DBLP), *mediální* (zejména zdroje BBC, pro které byly vyvinuty dedikované ontologie – např. Programmes Ontology nebo Wildlife Ontology), *biomedicínské* (řada genetických a dalších databází), *geografické* (např. GeoNames nebo World Factbook), ale také *veřejné správy*.

Datové zdroje *veřejné správy* mají v současnosti v rámci Linked Data zásadní postavení, jednak je zde možné se opřít o dlouhodobé vládní projekty (jako je nedávná iniciativa prezidenta Obamy za transparentní demokracii), jednak řada veřejných úřadů shromažďuje či přímo vytváří datové zdroje, které mohou v budoucnu sloužit jako „autority" pro propojování zdrojů jiných (tak jako to v současnosti platí o DBpedia) – jde zejména o číselníky publikované statistickými úřady a ministerstvy. Nejpokročilejší je stav v USA a Velké Británii, kde jsou data v otevřených formátech, vč. RDF, systematicky vystavována v rámci portálů http://www.data.gov/ a http://data.gov.uk/ spravovaných přímo vládami těchto států. V mnoha dalších zemích vznikají podobné portály cestou zdola (aktivitou nevládních organizací a akademických institucí); v ČR je to například http://opendata.cz/, který se v úvodní fázi soustřeďuje na informace o veřejných zakázkách.

Technologicky je iniciativa Linked Data v současnosti zabezpečena velkým počtem softwarových nástrojů, z nichž významná část je vzájemně propojitelná zejména v rámci architektury tzv. LOD2 Stack (vyvíjené v evropském projektu LOD2, http://lod2.eu).

2.5 Závěr

Sémantický web tak, jak byl prezentován na začátku kapitoly – tedy čistě strojové sloučení a zpracování velkého množství informací z heterogenních zdrojů – je stále ještě vizí. Zrovna tak ale můžeme říci, že sémantický web již je v určité podobě realizován. Při opětovném pohledu na architekturu sémantického webu z obr. 2.1 vidíme, že současné praktické aplikace se pohybují spíše na úrovni reprezentace dat v jazyce RDF či RDFa s použitím velmi jednoduchých slovníků a na využití těchto dat pomocí dotazování v jazyce SPARQL. Velmi povzbudivý je například nástup ontologie GoodRelations – zde se ukazuje, jak je pro využití v komerční sféře důležitá též správa a průběžná aktualizace existující ontologie.

Přestože bylo využívání ontologií ve webovém prostředí deklarováno jako hlavní směr sémantického webu od samotného počátku, ontologické inženýrství zůstávalo po dlouhou dobu zaměřeno na práci s jednotlivými, do značné míry monolitickými ontologiemi. Teprve v posledních několika letech převládl názor, že v sémantickém webu musí distribuovaně vznikat a mezi sebou se propojovat nejen dokumenty obsahující elementární data (fakta), ale i ontologie, které vymezují sémantiku těchto dat.[15] Lze předpokládat, že problémy spojené s nasazením ontologií v distribuovaném prostředí budou nadále významným tématem výzkumu; jde například o modularizaci větších ontologií, indexování ontologií pomocí metadat nebo znovupoužitelné návrhové vzory (těm se dále věnuje kapitola 4). Důležité je, nakolik se prokáže potřeba využívat sofistikovanější ontologie

[15] Výzkum takto pojatých ontologií byl předmětem nedávno skončeného evropského projektu *Networked Ontologies* (NeOn, http://www.neon-project.org/).

(oproti jednoduchým ad hoc slovníkům) ve spojení s rozsáhlými datovými zdroji Linked Data. Sémantický web se postupně stává „vícerychlostním" – na jedné straně je žádoucí dosáhnout vystavení co největšího rozsahu dat, na straně druhé ale předčasné propojování takových dat bez patřičné ontologické sémantiky nutně vede k řadě nekonzistencí, jejichž odstraňování vyžaduje nákladné ruční zásahy. Významnou roli zřejmě sehrají metody induktivního učení ontologií z datových zdrojů Linked Data, viz např. (Hellmann, 2009). V každém případě je další úspěšný rozvoj sémantického webu možný jen v úzké spolupráci mezi „ontologicky" a „datově" orientovanými částmi komunity.

Je vidět, že publikování dat z databází ve formátech sémantického webu začíná být alespoň po technologické stránce zvládnuté. I „jen" využití publikovaných datových zdrojů k lepším možnostem získávání informací je velkým posunem oproti čistě textovému webu, kde nemáme možnost slučovat data a pokládat dotazy v podobě složitějších grafových vzorů. Díky dostupným zdrojům nastává v současnosti návrat k původní myšlence sémantického webu – vlastnímu hlubšímu strojovému využití sémantiky v publikovaných informacích, tedy posunu od velkého distribuovaného grafu publikovaného v podobě RDF k využití dat pro získávání potřebných informací a znalostí. Jak vidíme na příkladech praktických aplikací, i jen zdánlivě triviální využití sémantiky – například hledání zboží v různých vzájemně nezávislých elektronických obchodech na základě částečné specifikace našich požadavků – znamená lepší a rychlejší nalezení toho, co doopravdy chceme.

Zájemcům o další studium v oblasti ontologií na sémantickém webu lze jako pokračování této kapitoly doporučit zejména: že pro praktický úvod do jazyka OWL spolu s editorem Protégé je k dispozici tutoriál (Horridge, 2009), obvyklé základní problémy a nedorozumění při tvorbě ontologií popisuje (Rector a kol., 2004), obecné shrnutí a využití sémantiky na příkladech ukazuje (Horrocks, 2008). Kapitoly 3 až 6 v této knize popisují do detailů některé vybrané partie spolu s doporučením dalších zdrojů. Jak již bylo řečeno, výzkum se nyní pohybuje zejména v prostředních vrstvách schématu znázorněného na obr. 2.1, příklady oblastí zájmu jsou například:

- kolaborativní tvorba a udržování větších ontologií v modulární a distribuované podobě, včetně publikování ontologií a dat na sémantickém webu,
- automatická tvorba ontologií z dat, z textů a dalších dokumentů na původním webu a automatické anotování již existujících dat a textů, zpracovávání dotazů v přirozeném jazyce pro dotazování v sémantickém webu,
- zpracování sémantiky a ontologií ve větším měřítku ve smyslu strojového odvozování, v distribuovaném, heterogenním prostředí s možnými nekonzistencemi, možnou nedostupností některých částí ontologie, s možnými změnami již publikovaných a použitých ontologií,
- využití pravidel pro odvozování dalších znalostí, pro integraci ontologií, posun existujících uzavřených pravidlových systémů a aplikací směrem k otevřenému webu,
- zajištění důvěry na otevřeném a distribuovaném sémantickém webu s velkým množstvím nedůvěryhodných zdrojů (tak jako na hypertextovém webu),
- vyhledávání ontologií a dat publikovaných pomocí těchto ontologií, mapování nalezených ontologií a dat pro další využití,
- bližší integrace myšlenek sémantického webu s fyzickým světem, tzv. „internet věcí" (Internet of Things), kde jsou jednoznačně identifikovány věci (v prvních aplikacích například senzory a další zařízení) tak, jako jsou na webu jednoznačně identifikovány

zdroje, a kde je umožněno integrovat a zpracovat jak informace o nich, tak data, která produkují,
- využití sémantického webu v sociálních sítích, například pro personalizaci a přizpůsobení se preferencím uživatele, spolu se zajištěním soukromí i přes zvýšenou dostupnost osobních informací,
- použití technologií sémantického webu nejen pro data, ale i pro webové služby, pro popis, integraci a řízení spouštění těchto služeb a předávání jejich výsledků,
- uživatelská rozhraní, která odstiňují uživatele od komplexnosti a rozsahu informací, a to nejen v textové podobě,
- a samozřejmě praktické aplikace, jež uvedené oblasti potřebují a využívají, které jsou schopny informace na sémantickém webu nalézat a využívat, včetně integrace sémantického webu do existujících aplikací pro využití jeho výhod.

Na těchto oblastech vidíme, že hlavní výzvou je zejména distribuované, heterogenní a stále se měnící prostředí obrovského rozměru. Některé z těchto oblastí umíme řešit v malém měřítku v neměnném prostředí a vidíme, jak i v tomto prostředí je řešení přínosné. Tak, jako si nikdo nedovedl představit dopad přechodu od izolovaného a konzistentního hypertextového systému do prostředí současného webu, tak můžeme jen uvažovat, co vše umožní strojové využití a zpracování všech informací, které na současném hypertextovém webu jsou. Doufejme, že myšlenky sémantického webu pomohou v realizaci inteligentního webu, kde hlavní nebudou dokumenty prezentované uživateli, ale kde hlavní roli bude mít poskytování nalezených a integrovaných informací, znalostí a služeb člověku dle jeho komplexních požadavků.

Literatura

Aduda B., Birbeck M., McCarron S., Pemberton S.: *RDFa in XHTML: Syntax and Processing*. W3C Recommendation, 2008. http://www.w3.org/TR/rdfa-syntax/

d'Aquin M. et al.: *Watson: A Gateway for the Semantic Web*. European Semantic Web Conference, ESWC, 2007.

Becket D., Broekstra J.: *SPARQL Query Results XML Format*. W3C Recommendation, 2008. http://www.w3.org/TR/rdf-sparql-XMLres/

Bersners-Lee T., Hendler J., Lassila O.: The Semantic Web. In: *Scientific American*, May 2001.

Biron P. V., Malhotra A.: *XML Schema Part 2*: Datatypes Second Edition. W3C Recommendation, 2004. http://www.w3.org/TR/xmlschema-2/

Brickley D., Guha R. V. (eds): *RDF Vocabulary Description Language 1.0*: RDF Schema, W3C Recommendation, 2004. http://www.w3.org/TR/rdf-schema/

Brickley D., Miller L.: *FOAF Vocabulary Specification 0.98*. Namespace Document, 2010. http://xmlns.com/foaf/spec/

Buitelaar P.: OntoSelect: Towards the Integration of an Ontology Library, Ontology Selection and Knowledge Markup (Position Paper). In: *Proc. of the Workshop on Knowledge Markup and Semantic Annotation (Semannot2004) at the International Semantic Web Conference*, Hirošima, Japonsko, November 2004.

Cohen W. W., Ravikumar P., Fienberg S. E.: A comparison of string distance metrics for name-matching tasks. In: *Proc. of the IJCAI-2003 Workshop on Information Integration on the Web* (IIWeb-03), 2003.

Connolly D. (ed.): *Gleaning Resource Descriptions from Dialects of Languages (GRDDL)*, W3C Recommendation, 2007. http://www.w3.org/TR/grddl/

David J., Euzenat J: On fixing semantic alignment evaluation measures. In: *Proc. of the ISWC 2008 Workshop on Ontology Matching*. 2008.

Li Ding et al.: „Swoogle: A Search and Metadata Engine for the Semantic Web". In: *Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management*. November 2004.

Euzenat J.: Semantic precision and recall for ontology alignment evaluation. In: *Proc. of 20th International Joint Conference on Artificial Intelligence* (IJCAI 07). 2007, s. 248–253.

Euzenat J., Shvaiko P.: *Ontology matching*. Springer, 2007.

van Hage W. R., Isaac A., Zharko A.: Sample evaluation of ontology matching systems. In: *Proc. of the ISWC 2007 workshop on Evaluation of Ontologies and Ontology-based tools*. 2007, s. 41–50.

Hayes P. (ed.): *RDF Semantics*. W3C Recommendation, 2004. http://www.w3.org/TR/rdf-mt/

Hellmann S., Lehmann J., Auer S.: Learning of OWL Class Descriptions on Very Large Knowledge Bases. In: *International Journal on Semantic Web and Information Systems*, 2009.

Hepp M., de Bruijn J.: GenTax: A Generic Methodology for Deriving OWL and RDF-S Ontologies from Hierarchical Classifications, Thesauri, and Inconsistent Taxonomies. In: *European Semantic Web Conference, ESWC*. LNCS 4519, Springer-Verlag, Innsbruck 2007, s. 129–144.

Horridge M.: *A Practical Guide to Building OWL Ontologies Using Protégé 4 and COODE Tools, edition 1.2*. The University of Manchester, 2009. http://owl.cs.manchester.ac.uk/ tutorials/protegeowltutorial/

Horridge M., Patel-Schneider P. F.: *OWL 2 Web Ontology Language Manchester Syntax*, W3C Working Group Note, 2009. http://www.w3.org/TR/owl2-manchester-syntax/

Horrocks I.: Ontologies and the semantic web, *Communications of ACM*, 51(12), 2008, s. 28–67.

Hu W., Qu Y.: Falcon-AO: A Practical Ontology Matching System, *Journal of Web Semantics*, 2007.

Manola F., Miller E. (eds): *RDF Primer*. W3C Recommendation, 2004. http://www.w3.org/TR/ rdf-primer/

McGuinness, D. L., van Harmelen, F. (eds): *OWL Web Ontology Language Overview*, W3C Recommendation, 2004. http://www.w3.org/TR/owl-features/

Meilicke C., Stuckenschmidt H.: Incoherence as a basis for measuring the quality of ontology mappings. In: *Proc. of the third ISWC 2008 workshop on Ontology Matching*, 2008.

Motik B., Patel-Schneider P. F., Grau B. C. (eds): *OWL 2 Web Ontology Language Direct Semantics*. W3C Recommendation, 2009. http://www.w3.org/TR/owl-direct-semantics/

Motik B., Patel-Schneider, P. F., Parsia, B. (eds): *OWL 2 Web Ontology Structural Specification and Functional-Style Syntax*. W3C Recommendation, 2009. http://www.w3.org/TR/owl2-syntax/

Patel-Schneider P. F., Motik B. (eds): *OWL 2 Web Ontology Language – Mapping to RDF Graphs*. W3C Reccomendation, 2009. http://www.w3.org/TR/owl-mapping-to-rdf/

Peroni S., Motta E., d'Aquin M.: Identifying key concepts in an ontology through the integration of cognitive principles with statistical and topological measures. In: *Third Asian Semantic Web Conference*. Bangkok 2008.

Prud'hommeaux E., Seaborne A. (eds): *SPARQL Query Language for RDF*. W3C Recommendation, 2008. http://www.w3.org/TR/rdf-sparql-query/

Rahm E., Bernstein P. A.: A survey of approaches to automatic schema matching. In: *The VLDB Journal*, Vol. 4, Issue 4, December 2001.

Rector A. et al.: *OWL Pizzas:* Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns, *Knowledge Acquisition, Modeling and Management (EKAW)*, 2004.

Schneider M. (ed.): *OWL 2 Web Ontology Language RDF-Based Semantics*. W3C Recommendation, 2009. http://www.w3.org/TR/owl-rdf-based-semantics/

Shvaiko P., Euzenat J: Ten Challenges for Ontology Matching. In: *Proc. of ODBASE* 2008.

Sirin E., Parsia B., Grau B. C., Kalyanpur A., Katz Y.: Pellet: A practical OWL-DL reasoner, *Journal of Web Semantics*, 5(2), 2007.

Sirin E., Parsia B.: SPARQL-DL: SPARQL Query for OWL-DL. In: *OWLED*, 2007.

Volz J., Bizer C., Gaedke M., Kobilarov G.: Silk – A Link Discovery Framework for the Web of Data. In: *Proc. of the 2nd Workshop about Linked Data on the Web (LDOW2009)*, Madrid, April 2009.

W3C OWL Working Group: *OWL 2 Web Ontology Language*. W3C Recommendation. 2009. http://www.w3.org/TR/owl-overview/

Yves R. Jean-Mary, Mansur R. Kabuka: ASMOV: results for OAEI 2008. In: *Proc. of the ISWC 2008 Workshop on Ontology Matching*, 2008.

3 Ontologie a deskripční logiky

3.1 Úvod

V předchozí kapitole se čtenář seznámil se základními aspekty sémantického webu a vybranými sémantickými technologiemi. Ty umožňují reprezentovat, sdílet a spravovat strukturované znalosti, a to od nejjednodušších slovníků, přes taxonomie, až po složité silně axiomatizované ontologické struktury snažící se věrně zachytit vztahy mezi prvky modelované doménové oblasti. V této kapitole se detailněji podíváme na rodinu jazyků, která dává logický základ mnohým z dříve diskutovaných technologií a systémů.

Na poli reprezentace znalostí se již od počátku střetávaly dva přístupy. Na jedné straně stojí jazyky založené na matematické logice. Tyto jazyky mají dobře definovanou deklarativní sémantiku (např. Prolog, KIF, OWL DL) a umožňují tak využít matematický aparát pro analýzu výpočetních vlastností a pro popis odvozovacích algoritmů. Druhou skupinu jazyků tvoří formalismy, jejichž sémantika je specifikována vágně (např. rámce, mapy témat apod.), které však umožňují vyjádřit znalosti, jež by pomocí aparátu matematické logiky nebylo možné modelovat, nebo by takové modelování bylo obtížné a často výpočetně nezvládnutelné. Typickým příkladem může být vyjadřování implicitních (default) hodnot atributů v rámcích popsané v kapitole 4 prvního dílu knihy (Mařík a kol., 1993) nebo modelování metaznalostí (znalosti o znalostech), pro které by byl často potřeba logický aparát vyššího řádu. Přes tato úskalí lze bez nadsázky říci, že formalismy založené na matematické logice jsou v současnosti dominantní v oblasti reprezentace znalostí a sémantického webu.

Již v osmdesátých a devadesátých letech minulého století byly studovány různé logické formalismy, a to v souvislosti s rozmachem expertních a znalostních systémů. Mnoho z těchto jazyků bychom dnes zařadili do rodiny formalismů, souhrnně nazývané *deskripční logiky*. Až na výjimky deskripční logiky za prvé odpovídají rozhodnutelným podmnožinám predikátové logiky prvního řádu, o níž je zmínka v kapitole 3 prvního dílu knihy, a za druhé slouží k popisu strukturálních vlastností dané domény. Jako takové dnes poskytují formální rámec standardům sémantického webu, zejména jazykům DAML+OIL, OWL nebo v současnosti nejbohatšímu jazyku OWL 2.

3.2 Deskripční logika *ALC*

Začátkem devadesátých let minulého století byl v článku (Schauss a Schmolka, 1991) představen jazyk *ALC* (z anglického výrazu *attributive language with complements*), který lze považovat za základ deskripčních logik v současnosti používaných na sémantickém webu. Fundamentálními prvky jazyka *ALC* jsou *koncepty*, které popisují množiny prvků dané domény, *role*, jež odpovídají binárním relacím mezi těmito prvky,

a *individuály*, kterými popisujeme prvky modelované domény. Různé deskripční logiky se pak od sebe vzájemně odlišují jednak sadou konstruktů, kterými lze tvořit složené koncepty a role, jednak druhy axiomů. Dále v textu budeme symboly $A_{(k)}$ označovat atomické (pojmenované) koncepty, symboly $R_{(k)}$ atomické role, $C_{(k)}$ libovolné koncepty a $i_{(k)}$ individuály.

3.2.1 Syntax a sémantika jazyka *ALC*

V jazyce *ALC* je možné tvořit složené koncepty (ovšem nikoliv složené role), a to pomocí tohoto gramatického pravidla

$$C \leftarrow A \mid \top \mid \bot \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \neg C \mid \exists R.C \mid \forall R.C. \tag{3.1}$$

Neformálně lze tyto konstrukty vysvětlit takto: druhý ($\top$) a třetí ($\bot$) konstrukt popisují univerzální a prázdnou množinu, další tři odpovídají klasickým množinovým operacím průniku, sjednocení a množinovému doplňku. Poslední dva konstrukty reprezentují množinu doménových prvků, které jsou v relaci R s nějakým prvkem patřícím do C, popř. množinu prvků, které nemusí být v relaci R s žádným doménovým prvkem, pokud ovšem jsou, musí to být prvek patřící do C.

Axiomy (někdy též formule) logiky *ALC* jsou čtyř typů, a to

- $C_1 \sqsubseteq C_2$ mající význam množinové inkluze,
- $C_1 \equiv C_2$ s významem ekvivalence množin.
- $C(i)$ označují příslušnost individuálu i ke konceptu C a
- $R(i_1, i_2)$, jež dává do vztahu R individuály i_1 a i_2.

Ontologie $\mathcal{K}$ deskripční logiky *ALC* je tvořena množinou axiomů rozdělenou na dvě části – tzv. TBox $\mathcal{T}$, který obsahuje všechny inkluze $C_1 \sqsubseteq C_2$ a ekvivalence $C_1 \equiv C_2$, a tzv. ABox $\mathcal{A}$, který obsahuje všechny axiomy $C(i)$ a $R(i_1, i_2)$.[1] Intuitivně, TBox reprezentuje zobecněnou, terminologickou znalost, zatímco ABox odpovídá konkrétní relační struktuře mezi objekty modelované domény.

Příklad. Pro názornější pochopení uvedených pojmů uvažujme, že bychom chtěli vytvořit informační systém pro správu genealogických dat (evidence rodokmenů, osob, jejich vztahů atd.) a jejich vyhledávání. Náš systém by přistupoval k vnějším databázím a informačním systémům (např. jednotlivých matrik), které mají rozdílné datové modely, a (tedy i) rozdílná vnější rozhraní. Hlavním problémem při tvorbě takovéhoto systému je právě integrace znalostí z více heterogenních zdrojů, pro kterou se dá s výhodou použít ontologií a aparátu deskripčních logik, viz (Baader a kol., 2003, kapitola 16). Pro začátek uvažujme atomické koncepty *Osoba*, *Muz*, *Prarodic* a atomickou roli *maDite* se zřejmým významem. Složený koncept označující ty osoby, které mají za své potomky pouze muže, nebo jsou zcela bez potomků, bychom zapsali výrazem $Osoba \sqcap \forall maDite.Muz$. Naopak složený koncept, který označuje ty osoby, které mají jak dcery, tak syny, popíšeme jako $Osoba \sqcap (\exists maDite.Muz) \sqcap (\exists maDite.(Osoba \sqcap \neg Muz))$.[2] Axiom, který definuje koncept *Prarodic*, by potom mohl vypadat třeba takto:

---

[1] V předchozí kapitole se čtenář seznámil s pojmem ontologie ve znalostním inženýrství. My jej zde budeme používat v poněkud užším významu, a to jako množinu logických axiomů.

[2] V nejednoznačných případech, tak jako v tomto, budeme používat závorky, které jsme pro přehlednost v gramatickém pravidle (3.1) vynechali.

$$Prarodic \equiv Osoba \sqcap (\exists maDite.\exists maDite\,.\top) \tag{3.2}$$

Podobně axiom, který tvrdí, že "JAN" je prarodičem, zapíšeme jako

$$Prarodic("JAN").$$

Formální sémantika deskripční logiky bude popsána v dalším odstavci, již nyní se však můžeme zamyslet nad tím, jak bychom vyjádřili tyto znalosti pomocí formulí predikátového kalkulu prvního řádu. Využijeme-li intuitivní korespondence atomických konceptů s unárními predikáty deskripční logiky a atomických rolí s predikáty binárními, můžeme první z uvedených axiomů přepsat pomocí této formule predikátového kalkulu (druhý z axiomů by odpovídal analogickému faktu predikátové logiky)

$$\forall x(Prarodic(x) \leftrightarrow (Osoba(x) \wedge \exists y\,(maDite(x, y) \wedge \exists z\; maDite(y, z)))).$$

Vidíme, že na rozdíl od predikátové logiky nevyužívá deskripční logika proměnné pro odkazování mezi odpovídajícími si částmi relačního popisu. Výsledkem je kompaktnější a čitelnější reprezentace. Tato vlastnost by však samozřejmě nestačila k tomu, aby se jazyk *ALC* stal základem logik, které se v současnosti těší takové oblibě. Jejich významnou výhodou proti predikátovému kalkulu je *rozhodnutelnost*, byť s vysokou výpočetní složitostí – úloha, zda daný (složený) koncept deskripční logiky *ALC* může modelovat neprázdnou množinu (tedy zda je splnitelný) je ve třídě PSPACE dokonce i v případě, kdy ontologie $\mathcal{K}$ neobsahuje žádné axiomy, viz (Baader a kol., 2003).

V předchozím odstavci jsme ukázali, jak vypadá popis stejného axiomu v deskripční logice a v predikátové logice prvního řádu. Podívejme se nyní podrobněji na sémantiku deskripční logiky. Pro ontologii $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ označíme množiny všech atomických konceptů, rolí a individuálů *NC*, *NR* a *NI*. Tyto tři množiny dohromady tvoří tzv. *signaturu* ontologie $\mathcal{K}$. K definici sémantiky využijeme klasickou interpretaci $I = (\Delta^I, \bullet^I)$ predikátového kalkulu prvního řádu (zavedenou v kapitole 4 prvního dílu svazku), kde Δ^I je (neprázdná) interpretační doména a $\bullet^I$ je interpretační funkce, která atomickému konceptu A přiřadí podmnožinu domény $A^I \subseteq \Delta^I$, atomické roli R binární relaci nad doménou $R^I \subseteq \Delta^I \times \Delta^I$ a individuálu i prvek domény $i^I \in \Delta^I$. Tabulka 3.1 uvádí rozšíření interpretační funkce $\bullet^I$ pro uvedené konstruktory konceptů.

Tabulka 3.1 Interpretace konstruktorů jazyka *ALC*

$\top^I$	=	Δ^I
$\bot^I$	=	$\emptyset$
$(C_1 \sqcap C_2)^I$	=	$C_1^I \cap C_2^I$
$(C_1 \sqcup C_2)^I$	=	$C_1^I \cup C_2^I$
$(\neg C)^I$	=	$\Delta^I \setminus C^I$
$(\exists R.C)^I$	=	$\{x \in \Delta^I \mid \exists y \langle x, y\rangle \in R^I \wedge y \in C^I\}$
$\forall\,(R.C)^I$	=	$\{x \in \Delta^I \mid \forall y \langle x, y\rangle \in R^I \rightarrow y \in C^I\}$

Mnoho dalších pojmů lze zavést zcela analogicky jako v predikátové logice. Zápisem $I \models \alpha$ budeme označovat platnost axiomu α v interpretaci I (takovou interpretaci nazveme též modelem axiomu α). Pro axiomy logiky *ALC* definuje relaci $\models$ tabulka 3.2.

Tabulka 3.2 Interpretace axiomů jazyka *ALC*

$I \models (C_1 \sqsubseteq C_2)$	když	$C_1^I \subseteq C_2^I$
$I \models (C_1 \equiv C_2)$	když	$C_1^I = C_2^I$
$I \models C(i)$	když	$i^I \in C^I$
$I \models R(i_1, i_2)$	když	$\langle i_1^I, i_2^I \rangle \in R^I$

Dále pro množinu axiomů $\boldsymbol{S}$ definujme $I \models \boldsymbol{S}$, pokud $I \models \alpha$ pro všechny axiomy α z $\boldsymbol{S}$. Axiom α je logickým důsledkem množiny axiomů $\boldsymbol{S}$ (označíme $\boldsymbol{S} \models \alpha$, platí-li $I \models \alpha$, vždy když $I \models \boldsymbol{S}$.

Příklad. Vraťme se na okamžik k našemu příkladu a uvažujme ontologii $\mathcal{K}_1 = (\{\alpha\}, \{\beta\})$, kde α, β jsou axiomy z rovnice (3.2), popř. (3.3). Interpretaci I_1, která má být modelem $\mathcal{K}_1$, můžeme nadefinovat např. takto:

- $\Delta^{I_1} = Muz^{I_1} = Osoba^{I_1} = \{Jan, Filip, Martin\}$,
- $maDite^{I_1} = \{\langle Jan, Filip\rangle, \langle Filip, Martin\rangle\}$,
- $Prarodic^{I_1} = \{Jan\}$,
- $"JAN"^{I_1} = Jan$.

Model uvedený v přechozím příkladu je konečný (má konečnou doménu) a ukazuje na důležitou vlastnost mnoha (ovšem nikoliv všech) deskripčních logik včetně *ALC*: má-li ontologie $\mathcal{K}$ model, potom má konečný model (finite model property). Druhou důležitou vlastností logiky *ALC* je, že každý složený koncept lze interpretovat relační strukturou ve tvaru kořenového stromu (tree model property). Obě tyto vlastnosti jsou velmi podstatné pro analýzu složitosti automatické dokazovací procedury a lze je pro logiku *ALC* snadno nahlédnout z průběhu tablových algoritmů představených v následujícím odstavci. Ačkoliv se vlastnost konečného modelu s vzrůstající expresivitou dané deskripční logiky (již např. pro logiku *ALCNI* představenou v oddílu 3.3) rychle ztrácí, jistá varianta vlastnosti stromového modelu je charakteristická i pro expresivní deskripční logiky a je základem jejich rozhodnutelnosti.

3.2.2 Inference, metody automatického dokazování

Nyní je čas podívat se na metody, jakým způsobem lze ověřovat logické důsledky pomocí metod automatického dokazování. V prvních terminologických systémech, např. (Borgida a kol., 1989), (Schmolze a kol., 1985), založených na aparátu deskripčních logik se kladl důraz na schopnost odpovídat dotazy týkající se TBoxu $\mathcal{T}$, zejména pak

- subsumpce – je koncept C_2 obecnější než C_1, neboli platí $\mathcal{T} \models C_1 \sqsubseteq C_2$?
- ekvivalence – reprezentují koncepty C_1 a C_2 stejné množiny, neboli platí $\mathcal{T} \models C_1 \sqsubseteq C_2$ a současně $\mathcal{T} \models C_2 \sqsubseteq C_1$?

- disjunkce – jsou interpretace konceptů C_1 a C_2 nutně disjunktní, neboli platí $\mathcal{T} \models C_1 \sqcap C_2 \sqsubseteq \bot$?
- nesplnitelnosti – je interpretace konceptu C vždy prázdná, neboli $\mathcal{T} \models C_1 \sqsubseteq \bot$?

Jednotlivé typy dotazů nejsou nezávislé, např. dotaz „Je koncept C_2 obecnější než C_1?" je ekvivalentní dotazu „Je $C_1 \sqcap \neg C_2$ nesplnitelné?". Podobně lze nalézt korespondence mezi ostatními typy dotazů, zejména pak redukcemi subsumpce, ekvivalence a disjunkce na splnitelnost. Je nutné zdůraznit, že tyto redukce lze provést pouze, umožňuje-li daný formalismus používat úplnou negaci $\neg C$ (jazyk *ALC*, stejně jako v dalším výkladu zmíněná logika *SHOIN*, tento předpoklad splňuje).

Podobně jako pro TBox i pro ABox $\mathcal{A}$ lze definovat několik základních typů dotazů:

- konzistenci (consistency checking) – je $\mathcal{A}$ konzistentní s $\mathcal{T}$, neboli existuje model $\mathcal{I}$, který je modelem $\mathcal{A}$?
- ověřování příslušnosti ke konceptu (instance checking) – je daný individuál i instancí konceptu C, neboli platí $\mathcal{T} \cup \mathcal{A} \models C(i)$?
- ověřování příslušnosti k roli (role checking) – je daný individuál i_1 spojen rolí R s instancí i_2, neboli $\mathcal{T} \cup \mathcal{A} \models R(i_1, i_2)$?
- získávání instancí (instance retrieval) – nalezni všechny individuály i, které jsou instancí konceptu C?
- realizace (realization) – pro daný individuál i a množinu konceptů nalezni nejspecifičtější koncept C z této množiny, pro nějž $\mathcal{T} \cup \mathcal{A} \models C(i)$.

Všechny tyto dotazy lze, podobně jako v případě TBoxu, redukovat na jeden nebo více dotazů testujících konzistenci nějakého ABoxu (ponecháme čtenáři jako cvičení). Posledním pozorováním je, že i problém nesplnitelnosti konceptu C lze redukovat na ověřování konzistence ABoxu $\mathcal{A} \cup \{C(i)\}$, kde i je nový individuál nevyskytující se v $\mathcal{A}$. V další sekci představíme algoritmus pro ověřování konzistence ABoxu, s tím že všechny předchozí dotazy lze s jeho pomocí (více či méně efektivně) zodpovědět. Důvodem, proč se tyto základní dotazy uvažují samostatně, jsou především velmi efektivní optimalizace, které jsou v inferenčních strojích pro deskripční logiku využívány. Čtenář může dobrý přehled těchto metod nalézt například v kapitole 9 knihy (Baader a kol., 2003).

Vzhledem k tomu, že jazyk *ALC* je podmnožinou predikátové logiky prvního řádu, lze pro automatické odvozování v logice *ALC* použít existující a ověřené algoritmy známé pro logiku predikátovou (např. rezoluční metodu, se kterou se čtenář seznámil v prvním dílu knihy). Jako v praxi výhodnější se však pro deskripční logiky v průběhu let ukázaly především tzv. *tablové algoritmy* (tableaux algorithms). Obecnější varianty těchto algoritmů byly již dříve známé pro predikátový kalkulus, rezoluční metoda se však pro predikátovou logiku prosadila lépe. Než se podíváme na tablové algoritmy blíže, doplňme ještě, že pro logiky slabší než *ALC* (např. jazyk *ALN*, viz (Baader, a kol., 2003)) byly vyvinuty dokonce časově polynomiální algoritmy k ověřování subsumpce konceptů založené na syntaktickém porovnání struktury obou (normalizovaných) stran

subsumpce. Tyto algoritmy však není možné jednoduše zobecnit pro logiky expresivnější, zejména pro ty, které umožňují modelovat úplnou negaci (konstruktor $\neg C$).

3.2.3 Tablové algoritmy

Hlavní myšlenku tablových algoritmů bychom mohli neformálně popsat asi takto: „Máme-li ověřit konzistenci dané množiny axiomů, zkusme sestrojit přímo její model. Pokud se nám to podaří, je tato množina axiomů konzistentní". V této sekci se pokusíme čtenářům nabídnout především intuitivní pohled do nitra tablových algoritmů spíše než jejich přísně formální popis; ten mohou zájemci nalézt pro jazyk *ALC* například v původním článku (Schmidt-Schauss M. a kol., 1991) a pro jeho rozšíření, dnes využívaná v oblasti sémantického webu, v publikaci (Horrocks a kol., 2005).

3.2.3.1 Grafy zúplnění

Pro reprezentaci možných interpretací se v průběhu tablového algoritmu využívají tzv. *grafy zúplnění* (completion graphs). Jde o datové struktury, ze kterých po skončení běhu tablového algoritmu snadno zkonstruujeme hledaný model, nebo vyloučíme jeho existenci, jestliže model neexistuje. *Stav* tablového algoritmu je tvořen množinou S takových grafů. Tablové algoritmy mění svůj stav pomocí předdefinované množiny *odvozovacích pravidel*. Množina těchto pravidel se liší pro každou deskripční logiku a představuje jádro celého odvozovacího algoritmu. Zaveďme nyní tyto pojmy trochu přesněji:

- *Graf zúplnění* je ohodnocený orientovaný graf $G = (V_G, E_G, \mathcal{L}_G)$, kde každý uzel $x \in V_G$ je ohodnocen množinou $\mathcal{L}_G(x)$ konceptů a každá hrana $\langle x, y\rangle \in E_G$ je ohodnocena množinou atomických rolí $\mathcal{L}_G(\langle x, y\rangle)$[3].
- Graf zúplnění G obsahuje *přímý spor*, pokud $\{A, \neg A\} \subseteq \mathcal{L}_G(x)$ nebo $\bot \in \mathcal{L}_G(x)$ pro nějaký vrchol x a nějaký atomický koncept A.
- Graf zúplnění G je *úplný*, pokud na něj nelze aplikovat žádné pravidlo z množiny pravidel daného tablového algoritmu[4].

Úplný graf zúplnění, který neobsahuje přímý spor tedy reprezentuje třídu modelů.

3.2.3.2 Tablový algoritmus pro jazyk ALC při prázdném TBoxu

Uvažujme konečnou ontologii $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ jazyka *ALC*, jejíž konzistenci budeme ověřovat. Pro začátek předpokládejme, že máme prázdný TBox, tedy $\mathcal{T} = \emptyset$.

Průběh tablového algoritmu

1. PŘEDZPRACOVÁNÍ. V prvním kroku je nutné transformovat všechny koncepty vyskytující se v $\mathcal{K}$ do tzv. *negační normální formy* (negational normal form, NNF). To znamená pomocí ekvivalentních úprav známých z výrokové a predikátové logiky (jejichž interpretace v kontextu deskripčních logik je přímočará a lze ji nechat čtenáři

[3] Přestože pro jazyk *ALC* bude množina $\mathcal{L}(\langle x, y\rangle)$ vždy jednoprvková, je vhodné si zavést zobrazení $\mathcal{L}$ právě takto – později nám to umožní snadno rozšířit zde zavedené pojmy nad rámec logiky *ALC*.

[4] Je třeba upozornit, že úplnost grafu zde používáme v jiném významu, než je obvyklé v teorii grafů (jako graf, ve kterém je hrana mezi každou dvojicí vrcholů).

jako jednoduché cvičení) přesunout negace k atomickým konceptům. Tato normalizace je nutná, abychom minimalizovali počet možností, z nichž může v průběhu tablového algoritmu vzniknout spor ve vyšetřovaných interpretacích. Na tomto místě je třeba čtenáře upozornit, že ačkoliv nám převod konceptů do negační normální formy výrazně zjednoduší popis tablového algoritmu, v praktických implementacích se tento převod většinou neprovádí a z důvodů efektivity se využívá jiných kanonických forem, viz (Baader a kol., 2003).

2. INICIALIZACE. Počáteční stav algoritmu je $S_0 = \{\{G_0\}\}$. V grafu G_0 vrcholy odpovídají všem individuálům vyskytujícím se v ABoxu. Pro každý vrchol i je jeho ohodnocení rovno množině všech konceptů C takových, že $C(i) \in \mathcal{A}$. Mezi dvěma vrcholy i a j je hrana (ohodnocená rolí R), pokud $R(i, j) \in \mathcal{A}$.

3. TEST KONZISTENCE. Označme si aktuální stav tablového algoritmu jako S. Pokud každý $G \in S$ obsahuje přímý spor, potom algoritmus končí s výsledkem NEKONZISTENTNÍ.

Tabulka 3.3 Odvozovací pravidla pro jazyk *ALC*

⊓-pravidlo	podmínka: $(C_1 \sqcap C_2) \in \mathcal{L}_G(i)$ a $\{C_1, C_2\} \not\subseteq \mathcal{L}_G(i)$ pro nějaké i akce: $S' = S \cup \{G'\} \setminus \{G\}$, kde $G' = (V_G, E_G, \mathcal{L}_{G'})$, $\mathcal{L}_{G'}(i) = \mathcal{L}_G(i) \cup \{C_1, C_2\}$, jinak se shoduje s $\mathcal{L}_G$ *„Je-li v ohodnocení vrcholu i koncept $(C_1 \sqcap C_2)$, ale nejsou v něm oba konjunkty C_1, C_2, potom je do tohoto vrcholu přidejme."*
⊔-pravidlo	podmínka: $(C_1 \sqcup C_2) \in \mathcal{L}_G(i)$ a $\{C_1, C_2\} \cap \mathcal{L}_G(i) = \emptyset$ pro nějaké i akce: $S' = S \cup \{G_1, G_2\} \setminus \{G\}$, kde $G_k = (V_G, E_G, \mathcal{L}_{G_k})$, $\mathcal{L}_{G_k}(i) = \mathcal{L}_G(i) \cup \{C_k\}$, $k \in \{1, 2\}$, jinak se shoduje s $\mathcal{L}_G$ *„Je-li v ohodnocení vrcholu i koncept $(C_1 \sqcap C_2)$, ale není v něm alespoň jeden z disjunktů C_1, C_2, potom tento graf zúplnění rozdělme na dva G_1, G_2 lišící se pouze v ohodnocení vrcholu x. V G_1 bude v ohodnocení vrcholu x navíc C_1 a v G_2 bude v ohodnocení vrcholu x navíc C_2."*
∃-pravidlo	podmínka: $(\exists R.C) \in \mathcal{L}_G(i)$ a neexistuje $j \in V_G$ takové, že $R \in \mathcal{L}_G(\langle i, j\rangle)$ a současně $C \in \mathcal{L}_G(j)$. akce: $S' = S \cup \{G'\} \setminus \{G\}$, kde $G' = (V_G \cup \{j\}, E_G \cup \{\langle i,j\rangle\}, \mathcal{L}_{G'})$, $\mathcal{L}_{G'}(j) = \{C\}$, $\mathcal{L}_{G'}(\langle i,j\rangle) = R$. jinak se shoduje s $\mathcal{L}_G$ *„Je-li v ohodnocení vrcholu i koncept $(\exists R . C)$ a současně není i spojen hranou ohodnocenou rolí R s nějakým vrcholem ohodnoceným konceptem C, potom vytvoříme nový vrchol, ohodnotíme jej konceptem C a spojíme i s ním hranou ohodnocenou rolí R".*
∀-pravidlo	podmínka: $(\forall R.C) \in \mathcal{L}_G(i)$ a existuje $j \in V_G$ takové, že $R \in \mathcal{L}_G(\langle i, j\rangle)$ a současně $C \notin \mathcal{L}_G(j)$. akce: $S' = S \cup \{G'\} \setminus \{G\}$, kde $G' = (V_G, E_G, \mathcal{L}_{G'})$, $\mathcal{L}_{G'}(j) = \mathcal{L}_G(j) \cup \{C\}$, jinak se shoduje s $\mathcal{L}_G$. *„Je-li v ohodnocení vrcholu i koncept $(\forall R.C)$ a současně je i spojen hranou ohodnocenou rolí R s nějakým vrcholem, který není ohodnocen konceptem C, potom tento vrchol konceptem C ohodnotíme".*

4. TEST MODELU. Vezmeme jeden $G \in S$, který neobsahuje přímý spor. Je-li G úplný vzhledem k pravidlům uvedeným v tabulce 3.3, potom algoritmus končí s výsledkem KONZISTENTNÍ.

5. APLIKACE PRAVIDLA. Nalezneme pravidlo, které je na G aplikovatelné. Aplikací tohoto pravidla získáme ze stavu S nový stav S' a algoritmus pokračuje bodem 2.

Korektnost

Korektnost představeného algoritmu je přímým a snadným důsledkem sémantiky jednotlivých odvozovacích pravidel. Dostaneme-li se ze stavu $S = \{G_k\}$ do stavu $S' = \{G_j\}$, potom musíme být schopni vytvořit model buď z obou těchto stavů, nebo ze žádného z nich. Například použitím $\sqcap$-pravidla víme, že měl-li stav S model I a pro nějaký vrchol i platilo $(C_1 \sqcap C_2) \in \mathcal{L}_{G_k}(i)$, potom $i^I \in (C_1 \sqcap C_2)^I$, a tedy dle tabulky 3.1, $i^I \in C_1^I$ a současně $i^I \in C_2^I$. Proto v G_j, který vznikl aplikací pravidla z G_k, platí $\{C_1, C_2\} \in \mathcal{L}_{G_k}(i)$. Pro opačný směr a ostatní pravidla lze úvahu provést analogicky.

Úplnost

Abychom ukázali úplnost uvedeného algoritmu, zkonstruujeme model pro každý úplný graf zúplnění, který neobsahuje přímý spor. Kanonický model I můžeme z grafu G vytvořit takto:

- doménou Δ^I budou všechny vrcholy grafu G,
- pro každý atomický koncept A definujeme $A^I = \{x \mid A \in \mathcal{L}_G(x)\}$,
- pro každou atomickou roli R definujeme $R^I = \{\langle x, y\rangle \mid R \in \mathcal{L}_G(\langle x, y\rangle)\}$.

Čtenář si může jako cvičení ověřit (indukcí podle struktury složených konceptů), že skutečně jde o model. V případě logiky ALC je korespondence mezi modelem a úplným grafem zúplnění zřejmá. Později uvidíme, že pro tablové algoritmy, které implementují některý z blokovacích mechanismů (viz odst. 3.2.3.3), může být tvorba modelu z grafu zúplnění poněkud obtížnější.

Konečnost

Konečnost algoritmu snadno nahlédneme z formy stavu tablového algoritmu:

- $\mathcal{K}$ je konečná.
- $\sqcup$-pravidlo jako jediné obohacuje stav o nový graf zúplnění, a to pouze jeden.
- Po aplikaci libovolného deterministického pravidla ($\sqcap$, $\exists$, $\forall$) se buď vytvoří nový vrchol, nová hrana, nebo se přidá ohodnocení k vrcholu nebo k hraně. Všechny tyto operace jsou konečné.
- Každý graf zúplnění může obsahovat nejvýše tolik vrcholů, kolik je existenciálních kvantifikátorů v $\mathcal{A}$ plus počet individuálů v $\mathcal{A}$.

Z této jednoduché analýzy je zřejmé, že uvedený algoritmus musí skončit po konečném počtu kroků (aplikací inferenčních pravidel).

Pečlivý čtenář může oprávněně nabýt dojmu, že graf zúplnění je vlastně pouze jinou vizualizací ABoxu. Jsou v zásadě dva důvody, proč zde pojem grafu zúplnění využíváme: jednak se grafová reprezentace lépe hodí pro vysvětlení průběhu tablových algoritmů, neboť názorněji přiblíží strukturu tvořeného modelu, především však se kore-

spondence mezi ABoxem a grafem zúplnění vytrácí v případě tablových algoritmů pro jazyky expresivnější než *ALC*.

Na tablové algoritmy lze nahlížet jako na formu produkčních systémů, zmíněných v prvním díle tohoto svazku. S tím souvisí i problémy, které je nutné při návrhu a implementaci tablového algoritmu řešit. Jedná se zejména o řešení konfliktů mezi pravidly (jaké pravidlo vybrat), mezi částmi grafu (pravidlo může být aplikovatelné na různé vrcholy) a mezi jednotlivými grafy (který bezesporný graf vybrat pro další expanzi). Tyto problémy lze řešit s využitím různých heuristik a přesahují rámec tohoto přehledového textu o deskripčních logikách. Zájemci mohou nalézt více informací např. v kapitole 9 knihy (Baader a kol, 2003).

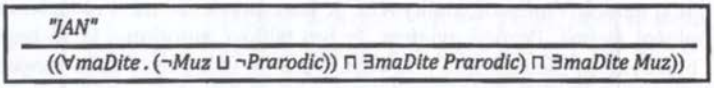
Příklad. Mějme ontologii $\mathcal{K}_2 = (\emptyset, \{\gamma\})$, jejíž konzistenci bychom chtěli otestovat, kde γ je axiom

$$(\exists maDite.Muz \sqcap \exists maDite.Prarodic \sqcap \exists maDite.(Muz \sqcap Prarodic))(\text{"}JAN\text{"}). \quad (3.3)$$

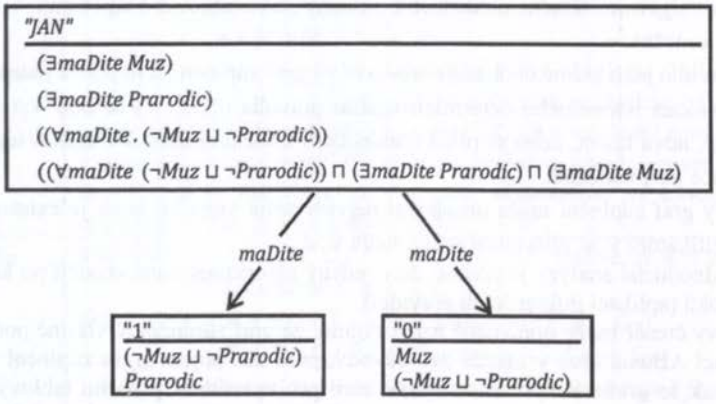
Prvním krokem je převod do negační normální formy. V našem případě využijeme nejprve ekvivalence konceptů $(\neg\exists R.C)$ a $(\forall R.\neg C)$, poté De Morganových pravidel, a dostáváme γ' v NNF ve tvaru

$$(\exists maDite.Muz \sqcap \exists maDite.Prarodic \sqcap \forall maDite.\ (\neg Muz \sqcap \neg Prarodic))(\text{"}JAN\text{"}). \quad (3.4)$$

Axiom (3.4) je ekvivalentní s axiomem (3.5) a má negace pouze před atomickými koncepty. Počáteční stav G_0 tablového algoritmu je znázorněn na obrázku 3.1.



Obr. 3.1 Počáteční stav tablového algoritmu.



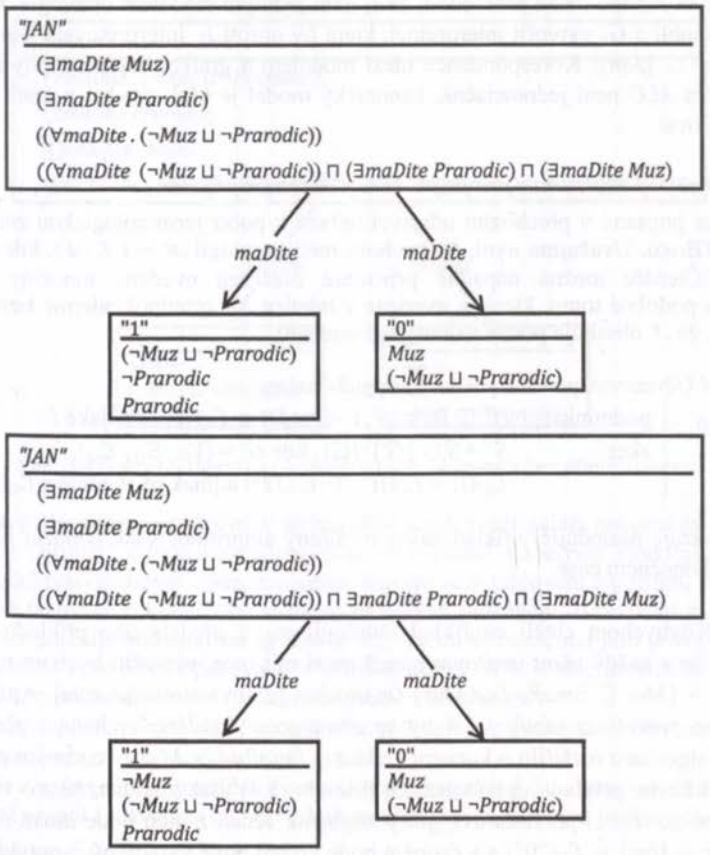
Obr. 3.2 Stav $\{G_4\}$ po aplikaci deterministických pravidel.

V tomto okamžiku algoritmus provede 4 sekvence kroků 2, 3, 4, které můžeme znázornit vývojem stavu (nad šipkou je znázorněno použité pravidlo) během kroku 4:

$$\{G_0\} \xrightarrow{\sqcap\text{-pravidlo}} \{G_1\} \xrightarrow{\exists\text{-pravidlo}} \{G_2\} \xrightarrow{\exists\text{-pravidlo}} \{G_3\} \xrightarrow{\forall\text{-pravidlo}} \{G_4\},$$

kde G_4 je znázorněn na obrázku 3.2.

Až dosud se v průběhu algoritmu aplikovala pouze deterministická pravidla (tedy pravidla, která nezvyšují počet vyšetřovaných grafů zúplnění). Podíváme-li se pozorně na obrázek 3.2, vidíme, že nyní můžeme použít již pouze (nedeterministické) $\sqcup$-pravidlo. Toto pravidlo lze použít na koncept $(\neg Muz \sqcup \neg Prarodic)$, a to buď v ohodnocení vrcholu "0", nebo vrcholu "1". Zvolme si kupříkladu vrchol "1". Po aplikaci pravidla získáme stav $\{G_5, G_6\}$.



Obr. 3.3 Stav $\{G_5, G_6\}$ po aplikaci nedeterministického pravidla. G_5 je v horní části, G_6 v dolní.

Vidíme, že graf G_5 obsahuje přímý spor ve vrcholu "1" a nebude již tedy dále vyšetřován. Každý model, který by bylo možné zkonstruovat z tohoto grafu zúplnění by nutně vyžadoval, aby objekt "1" patřil současně do interpretace *Prarodic* a $\neg$*Prarodic*. Zbývá tedy vyšetřit graf G_6. Po aplikaci $\sqcup$-pravidla (které je opět jediné aplikovatelné) získáme stav $\{G_5, G_7, G_8\}$, kde grafy G_7, G_8 vznikly z G_6.

Vidíme, že graf G_8 obsahuje přímý spor ve vrcholu "0", a nebude již dále vyšetřován. Graf G_7 je úplný (nelze aplikovat žádná další pravidla) a neobsahuje přímý spor, algoritmus tedy skončí v bodě *3.* Z grafu G_7 již můžeme přímo zkonstruovat model I_2:

- $\Delta^{I_2} = \{Jan, i_1, i_2\}$,
- $maDite^{I_2} = \{\langle Jan, i_1\rangle, \langle Jan, i_2\rangle\}$,
- $Prarodic^{I_2} = \{i_1\}$,
- $Muz^{I_2} = \{i_2\}$,
- $"JAN"^{I_2} = Jan, "0"^{I_2} = i_2, "1"^{I_2} = i_1$.

Čtenář se snadno může přesvědčit, že I_2 není jediným modelem ontologie $\mathcal{K}_2$. Stejně tak jsme mohli z G_7 vytvořit interpretaci, která by oproti I_2 interpretovala koncept *Muz* jako $Muz^{I_2} \cup \{Jan\}$. Korespondence mezi modelem a grafem zúplnění tedy ani v případě jazyka *ALC* není jednoznačná, kanonický model je však snadné z grafu zúplnění zrekonstruovat.

3.2.3.3 Tablový algoritmus pro jazyk ALC v obecném případu

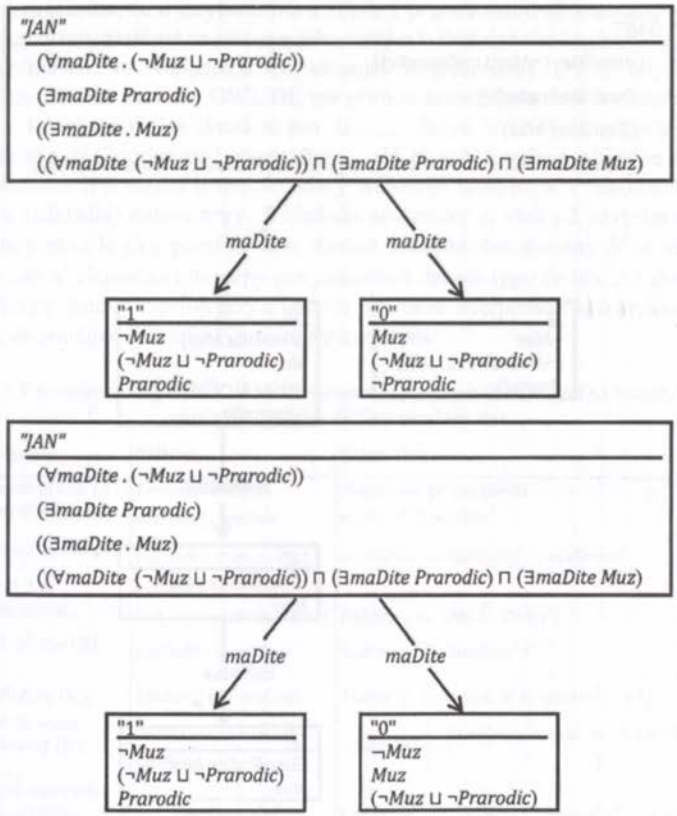
Algoritmus popsaný v přechozím odstavci nebere v potaz terminologickou znalost uloženou v TBoxu. Uvažujme nyní, že bychom měli ontologii $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, kde $\mathcal{T}$ je neprázdný. Čtenáře možná napadne přirozené rozšíření uvedené množiny pravidel o pravidlo podobné tomu, které je uvedené v tabulce 3.4 (předpokládejme bez újmy na obecnosti, že $\mathcal{T}$ obsahuje pouze axiomy subsumpce).

Tabulka 3.4 Odvozovací pravidlo pro terminologické axiomy

$\sqsubseteq$-pravidlo	podmínka:	$(C \sqsubseteq D) \in \mathcal{T}, (\neg C \sqcup D) \notin \mathcal{L}_G(i)$ pro nějaké i
	akce:	$S' = S \cup \{G'\} \setminus \{G\}$, kde $G' = (V_G, E_G, \mathcal{L}_{G'})$, $\mathcal{L}_{G'}(i) = \mathcal{L}_G(i) \cup (\neg C \sqcup D)$ a jinak se shoduje s $\mathcal{L}_G(i)$.

Jak ukazuje následující příklad, takto rozšířený algoritmus však bohužel již nemusí skončit v konečném čase.

Příklad. Kdybychom chtěli například ontologii $\mathcal{K}_2$ z předchozího příkladu obohatit o tvrzení, že v každý námi uvažovaný muž musí mít otce, vytvořili bychom neprázdný TBox $\mathcal{T}_2 = \{Muz \sqsubseteq \exists maRodice.Muz\}$ (*maRodice* je nová atomická role). Aplikací odvozovacího pravidla z tabulky 3.4 by se ohodnocení každého vrcholu v předchozím tablovém algoritmu rozšířilo o koncept $(\neg Muz \sqcup \exists maRodice.Muz)$. Pozorným pohledem na (v předchozím příkladu úplný) graf G_7 (viz obr. 3.4) však zjistíme, že pro vrchol "0" získáme po použití $\sqcup$-pravidla dva grafy zúplnění. Jeden z nich bude ihned obsahovat spor $\{Muz, \neg Muz\} \in \mathcal{L}_G("0")$ a v druhém bude vrchol nově vytvořený $\exists$-pravidlem opět ohodnocen konceptem *Muz*, a uvedený algoritmus znovu vygeneruje nový vrchol atd., viz obrázek 3.5.



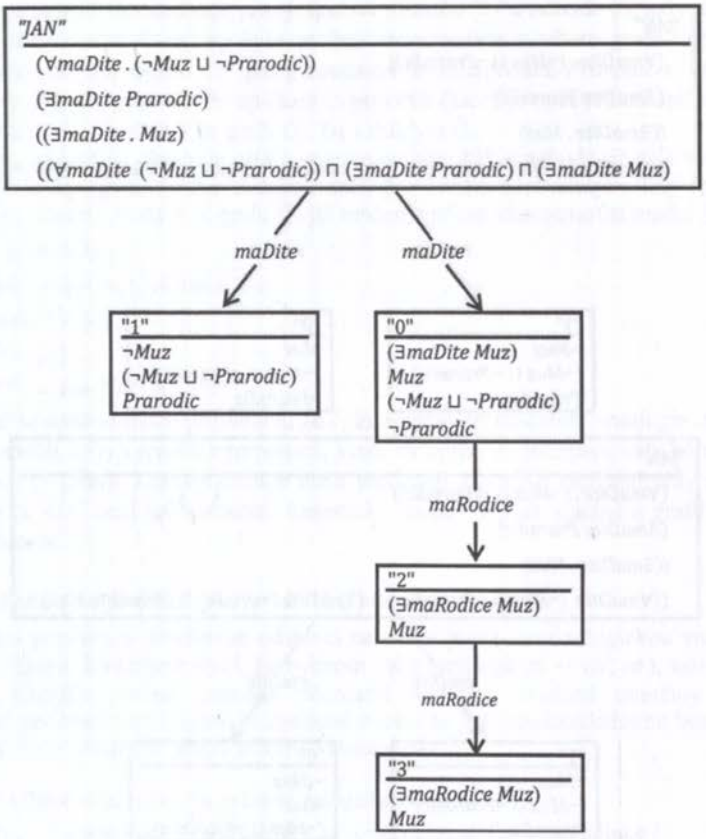
Obr. 3.4 Grafy G_7 (horní) a G_8 (dolní). G_7 je již úplný a bez přímého sporu.

Tablový algoritmus se vlastně v těchto příkladech snaží nalézt nekonečný model. Je tedy třeba „přimět" odvozovací algoritmus, aby hledal konečnou strukturu, ze které půjde model rekonstruovat[5]. Mechanismus, kterým se v tablovém algoritmu vynutí hledání právě takovýchto konečných struktur se nazývá *blokování*. Blokování zajišťuje, že odvozovací pravidla bude možné aplikovat jen, dokud se nezačnou jimi prováděné změny „dostatečně pravidelně opakovat". V případě logiky *ALC* je toto pravidelné opakování určeno tzv. *podmnožinovým blokováním* (subset blocking):

V grafu úplnění G je *vrchol x (nevyskytující se v $\mathcal{A}$) blokován vrcholem y*, vede-li orientovaná cesta od y k x a $\mathcal{L}_G(x) \subseteq \mathcal{L}_G(y)$.

Každé z inferenčních pravidel uvedených v tabulkách 3.3 a 3.4 je možné aplikovat jen, pokud vrchol i v uvedených pravidlech není blokován nějakým jiným vrcholem.

[5] Vzhledem k tomu, že každá konzistentní ontologie *ALC* má konečný model, bude takto nalezená struktura (graf úplnění) triviálně odpovídat hledanému konečnému modelu. V případě jazyků, které nemají vlastnost konečného modelu, je nutné nalezenou konečnou strukturu nejprve na (nekonečný) model transformovat.



Obr. 3.5 Konečnost tablového algoritmu a blokování.

Takto upravený tablový algoritmus je již rozhodovací procedurou pro libovolnou teorii jazyka *ALC* pracující ve třídě EXPTIME. O korektnosti, úplnosti a konečnosti takto rozšířeného algoritmu není obtížné se přesvědčit, zájemci mohou nalézt příslušný rozbor v knize (Baader a kol., 2003).

3.3 Expresivní deskripční logiky

Až dosud jsme se zabývali deskripční logikou *ALC*, jejíž vyjadřovací schopnosti by však samy o sobě patrně nemohly být základem pro ontologické technologie a sémantický web. Jazyk *ALC* se stal základem pro expresivnější jazyky, jejichž výpočetní vlastnosti byly v nedávné době velmi intenzivně zkoumány, viz (Horrocks a kol., 1998) a (Horrocks a kol., 2004). Tyto expresivnější jazyky pak položily formální rámec pro jazyk OWL (Web Ontology Language) standardizovaný konzorciem W3C v roce 2004 a rovněž jeho rozšíření OWL 2, které bylo standardizované v roce 2009.

Vzhledem k tomu, že o jazyku OWL a OWL 2 je podrobněji diskutováno v předcházejících kapitolách, zmiňme se pouze o jeho vztahu k deskripčním logikám. Jazyk OWL je standardizován ve variantách (s klesající expresivitou) OWL Full, OWL DL a OWL Lite, přičemž varianta OWL DL má přímou korespondenci s deskripční logikou *SHOIN(D)*, kde symbol *S* je zkratkou pro ALC_{trans}. Jazyk *SHOIN(D)* tedy vznikne obohacením jazyka *ALC* o výrazové prostředky $_{\text{trans}}$, *H*, *O*, *I*, *N* uvedené v tabulce 3.5.

Za zmínku stojí poslední konstrukt, který umožňuje modelovat z praktického hlediska důležité (základní) datové typy. Z hlediska sémantiky si však již nevystačíme s klasickou interpretací logiky prvního řádu. Kromě interpretační domény Δ^I je třeba navíc definovat (od Δ^I disjunktní) domény pro jednotlivé datové typy (v tab. 3.5 doména D^{int} pro celočíselný datový typ integer) a upravit příslušné interpretace konstruktorů jazyka v případě, že použijeme datový typ na místě konceptu.

Tabulka 3.5 Konstruktory jazyka OWL. Ve sloupci „Typ–popis" označuje (A) axiom, (K) konstruktor konceptu, (R) konstruktor role

	Typ–popis	**Příklad**	**Sémantika**
trans	Tranzitivní role (A)	*Trans(skladaSeZ)*	$skladaSeZ^I$ je tranzitivní
H	Hierarchie rolí (A)	$maSyna \sqsubseteq maDite$	$maSyna^I \subseteq maDite^I$
R	Kompozice rolí (A)	$maRodice \circ maBratra \sqsubseteq maStryce$	$maRodice^I \circ maBratra^I \subseteq maStryce^I$
O	Nominály (K)	$Pohlavi \equiv \{MUZ, ZENA\}$	$Pohlavi^I \equiv \{MUZ^I, ZENA^I\}$
I	Inverzní role (R)	$maRodice \sqsubseteq maDite^{-1}$	$maRodice^I \sqsubseteq (maDite^I)^{-1}$
N	Kardinalita (K)	$Matka \sqsubseteq (\geq 1\ maDite)$	$Matka^I \subseteq \{x \mid \lvert\{ y \mid \langle x, y\rangle \in maDite^I\}\rvert \geq 1\}$
Q	Kvalifikované kardinality (K)	$Matka \sqsubseteq (\geq 1\ maDite\ .\ Osoba)$	$Matka^I \subseteq \left\{ x \middle\vert \lvert\{ y \mid \langle x, y\rangle \in maDite^I \wedge y \in Osoba^I\}\rvert \geq 1 \right\}$
(D)	Použití datových typů na pozici konceptů.	$\top \sqsubseteq \forall maPocetKol.int$	$\forall x\ (x \in \Delta^I \wedge \forall y \langle x, y\rangle \in maPocetKol^I \rightarrow y \in D^{\text{int}})$

Jazyk OWL 2 je rozšířením jazyka OWL. Jeho rozhodnutelná varianta OWL 2 DL je rozšířením OWL DL (a tedy deskripční logiky *SHOIN(D)* o konstrukty *R*, *Q* uvedené v tab. 3.5 – výsledná deskripční logika se označuje *SROIQ(D)*. Jazyk OWL 2 nedefinuje analogii (stále exponenciální) varianty OWL Lite, ovšem obsahuje definici tří profilů (podmnožin konstruktů) OWL 2 QL, OWL 2 EL a OWL 2 RL vhodných pro různé úlohy. Jejich popis nalezne čtenář v předchozí kapitole svazku. My zde pouze konstatujeme, že OWL 2 EL a OWL 2 RL poskytují inferenční procedury, jejichž složitost je polynomiální (PTIME). Profil OWL 2 QL má složitost stejnou jako relační databáze, poskytuje však oproti nim striktně větší expresivitu (např. schopnost reprezentovat nekonečné modely). Zvýšení expresivity jazyka však s sebou přináší též komplikace při tvorbě rozhodovací procedury. Tablový algoritmus pro jazyk *ALC* uvedený v odst. 3.2.3.2 již samozřejmě pro *SHOIN(D)* ani *SROIQ(D)* není úplný a je třeba jej v několika ohledech upravit. Nebudeme zde uvádět úplnou formulaci rozhodovacích procedur pro tyto logiku (zájem-

ce odkažme na (Horrocks a kol., 2005)) a spokojíme se jen s neformálním popisem nejdůležitějších úprav popsaného tablového algoritmu:

- přidání pravidel pro nové konstrukty jazyka ($_{\text{trans}}$, O, N, Q),
- definicí R-okolí vrcholu (sousedé v relaci R, nebo v relaci sémanticky ekvivalentní) a příslušné zobecnění stávajících pravidel (díky H, I, R),
- přidáním podmínek pro ukončení inference pro detekci přímého sporu,
- rozšířením blokovacích podmínek.

Důležitým rozdílem je též forma grafů zúplnění. Zatímco v případě tablového algoritmu pro jazyk ALC jsme nalezli přímou souvislost mezi úplným grafem zúplnění a modelem, pro jazyk $SHOIN(D)$ a $SROIQ(D)$ již je tato souvislost netriviální. Důvod je ten, že tyto jazyky již nemají vlastnost konečného modelu, a tedy (konečné) grafy zúplnění mohou odpovídat pouze modelům nekonečným. Jako důsledek se problém konzistence posouvá ze třídy složitosti EXPTIME pro jazyk ALC do třídy složitosti NEXPTIME pro jazyk $ALC_{\text{trans}}HOIN(D)$ a do třídy N2EXPTIME pro jazyk $ALC_{\text{trans}}ROIQ(D)$, viz např. (Zolin, 2007).

3.4 Pokročilé inferenční procedury

V předchozích odstavcích jsme čtenáři nabídli stručný pohled do nitra deskripčních logik spolu s metodami pro automatické dokazování. Je zřejmé, že pro praktickou práci se znalostmi v deskripčních logikách by pouhá rozhodovací procedura pro určování konzistence byla poněkud těžkopádným nástrojem. V této kapitole představíme dvě (z praktického hlediska velmi užitečné) služby, které jsou nabízeny systémy založenými na deskripční logice a které staví na popsaných tablových algoritmech – vysvětlování modelovacích chyb a vyhodnocování konjunktivních dotazů.

3.4.1 Vysvětlování modelovacích chyb

Představme si, že vytváříme ontologii $\mathcal{K}$ deskripční logiky a v určitém okamžiku chceme zkontrolovat správnost našeho návrhu. Inferenční stroj vyhodnotí jeden z námi modelovaných konceptů jako nesplnitelný. Ve většině případů jde o modelovací chybu (modelovali bychom prázdnou množinu) a my stojíme před problémem, jak tuto nesplnitelnost odstranit. Obsahuje-li $\mathcal{K}$ řádově desítky axiomů, můžeme postupně tyto axiomy zkontrolovat a patrně se nám podaří nalézt a opravit ty chybné poměrně rychle. Ovšem již pro ontologii o řádově stovkách axiomů bude toto hledání zdlouhavé a pro ontologie obsahující řádově desetitisíce až statisíce axiomů zcela nemožné.

Nesplnitelnost konceptu není jediným problémem, pro který se hodí vysvětlovací techniky. Mezi jejich další aplikace patří vysvětlování nekonzistence ontologie $\mathcal{K}$, popř. obecně vysvětlování libovolného logického důsledku plynoucího z $\mathcal{K}$. Lze ukázat (Kalyanpur a kol., 2007), že tyto problémy jsou na sebe navzájem převeditelné a my se proto budeme zabývat bez újmy na obecnosti pouze problémem vysvětlování nesplnitelnosti konceptů.

Z uvedeného výkladu by mělo být zřejmé, že právě problém vysvětlování modelovacích chyb v ontologiích je při vývoji ontologie velmi zásadní. Široce přijatou forma-

lizací pro vysvětlování chyb se staly *minimální množiny axiomů* (ve smyslu množinové inkluze), které danou modelovací chybu zapříčiňují. Pro tyto množiny se v případě vysvětlování nesplnitelnosti daného konceptu vžilo označení MUPS (minimal unsatisfiability preserving subterminology).

Pro ontologii $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ a atomický koncept A označíme jako MUPS takovou množinu $M \subseteq \mathcal{T} \cup \mathcal{A}$ pro kterou $M \models (A \sqsubseteq \bot)$ a která je minimální s touto vlastností ve smyslu množinové inkluze.

Příklad. Uvažujme následující ontologii $\mathcal{K}_3 = (\{\alpha_1, \alpha_2, \alpha_3\}, \{\})$:

(α_1) $Osoba \sqsubseteq \exists maRodice.\ (Muz \sqcap Zena) \sqcap \forall maRodice.\ (\neg Osoba)$

(α_2) $Muz \sqsubseteq \neg Zena$

(α_3) $Muz \sqcup Zena \sqsubseteq Osoba$

Zatímco α_2, α_3 se zdají být v pořádku, α_1 je již na první pohled chybné. Čtenář si může snadno ověřit, že nesplnitelnost konceptu *Osoba* plyne nezávisle ze dvou množin axiomů, a to $\{\alpha_1, \alpha_2\}$ a $\{\alpha_1, \alpha_3\}$. Každá z těchto množin je MUPS.

Otázkou je, jak automaticky zkonstruovat jednu nebo všechny tyto množiny. V zásadě existují dva možné přístupy:[6]

- **Metody black-box** jsou takové, které nezasahují do běhu inferenčního stroje, pouze jej využívají pro vyhodnocení mnoha testů konzistence, na základě kterých zkonstruují jednotlivé MUPS. Tyto metody jsou nezávislé na zvoleném formalismu a jsou tedy univerzálně přepoužitelné pro různé typy úloh.
- **Metody glass-box** jsou integrovány do inferenčního stroje. Hlavní výhodou těchto metod je rychlost, jejich přepoužitelnost pro jiný inferenční stroj je však omezená.

3.4.1.1 *Metody Glass-box*

Algoritmus Glass-box pro vysvětlování modelovacích chyb v logice *ALC* lze nalézt v technické zprávě (Schlobach a kol., 2006). Původní tablový algoritmus pro ověřování konzistence logiky *ALC* je rozšířen o paměť axiomů, které byly použity při konstrukci sporných grafů zúplnění. Tyto množiny axiomů pak lze snadno převést na MUPS. Ač existují jeho rozšíření na mírně expresivnější deskripční logiky, není bohužel stále známo jeho zobecnění pro jazyky odpovídající současným trendům sémantického webu OWL, popř. OWL 2. Tento problém je řešen tím, že se metody glass-box pro méně expresivní jazyk (např. *ALC*) použijí pouze jako předzpracování před vlastním hledáním vysvětlení pomocí metod black-box.

3.4.1.2 *Metody Black-box*

Z prostorových důvodů se zde budeme podrobněji zabývat pouze algoritmy black-box, které nejsou striktně vázány na kalkulus deskripčních logik a lze je tedy snadno použít pro vysvětlování modelovacích chyb i v jiných (monotónních) formalismech. Obecně lze říci, že problém nalezení jednoho (libovolného) MUPS je řešitelný v polynomiálním čase vzhledem k počtu interakcí s inferenčním strojem, zatímco problém nalezení všech MUPS je již exponenciálně složitý.[7]

---

[6] Z důvodů absence odpovídající české terminologie si dovolíme použít v anglické literatuře dobře zavedené pojmy metody glass-box a black-box.

[7] Čtenář si jistě z předchozích odstavců vzpomíná, že však již samotný problém konzistence má u většiny desktipčních logik složitost v čase exponenciální nebo horší.

Algoritmy založené na CS-stromech

Hlavní myšlenkou algoritmů založených na *CS-stromech* (conflict set tree, CS-tree) je snaha o systematizaci hledání MUPS a minimalizaci počtu (typicky časově náročných) testů konzistence. Každý CS-strom je vlastně reprezentací stavového prostoru, jehož každý stav s má tvar (D, P), kde D obsahuje množinu axiomů, které *nutně musí být* ve všech MUPS nalezených v podstromu s kořenem ve vrcholu s, a P obsahuje množinu axiomů, které *mohou být* v některých MUPS nalezených v podstromu s kořenem ve vrcholu s.

Pro nesplnitelný koncept A a ontologii $\mathcal{K}$ pak základní algoritmus založený na CS-stromech je pouze modifikací prohledávání do hloubky s prořezáváním uzlů, v nichž je A nutně splnitelné (využívá se zde monotónosti daného logického kalkulu):

1. INICIALIZACE. Kořenem stromu je počáteční stav $s_0 = (\{\}, \mathcal{K})$, který reprezentuje fakt, že apriori neznáme axiomy, které musí být nutně v každém hledaném MUPS ($D_{s_0} = \{\}$), ovšem mohou tam být všechny ($P_{s_0} = \mathcal{K}$), dále položme $Z = (s_0)$ a $R = \{\}$.

2. PROHLEDÁVÁNÍ DO HLOUBKY. Je-li Z prázdný, ukončíme prohledávání. Jinak vyjmeme první prvek ze Z a označíme jej s.

3. TEST. Neplatí-li $D_s \cup P_s \models (A \sqsubseteq \bot)$, nemůže ani z žádné podmnožiny $D_s \cup P_s$ plynout nesplnitelnost A a není třeba vyšetřovat potomky s – pokračujeme prohledáváním grafu v bodě 2.

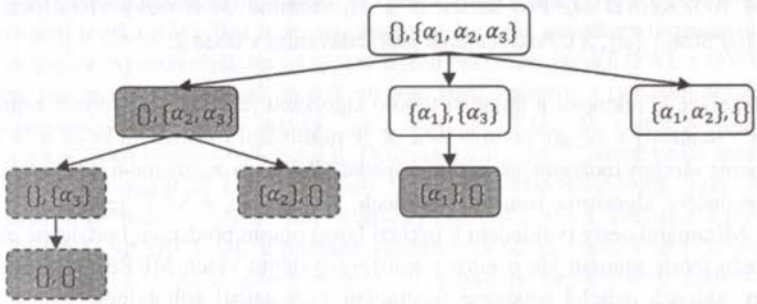
4. NALEZENÍ NESPLNITELNÉ MNOŽINY. Přidáme množinu $D_s \cup P_s$ do R a naopak odstraníme z R všechny s' takové, že $D_s \cup P_s \subseteq s'$.

5. EXPANZE STROMU. Nechť $P_s = \{\alpha_1, \ldots, \alpha_N\}$. Pro každé i takové, že $N \geq i \geq 1$ přidáme na začátek zásobníku Z nový uzel $(D_s \cup \{\alpha_1, \ldots, \alpha_{i-1}\}, P_s \setminus \{\alpha_1, \ldots, \alpha_i\})$. Pokračujeme v prohledávání grafu v bodě 2.

Po provedení tohoto algoritmu obsahuje R hledané MUPS. *Korektnost* a *úplnost* algoritmu je zřejmá ze způsobu generování nových uzlů v bodě 5. Pro každého potomka s' uzlu s je $D_{s'} \cup P_{s'} \subseteq D_s \cup P_s$ a nově vygenerované vrcholy pokrývají všechny varianty MUPS, které mohou nastat. *Konečnost* algoritmu je přímým důsledkem konečnosti vstupů a zmenšování $|D_s \cup P_s|$ s hloubkou stromu.

Tento základní algoritmus lze optimalizovat pro případ, že máme dodatečnou informaci o struktuře hledaných MUPS, např. nezávislosti axiomů *(axiomy, které nemohou patřit do stejného MUPS).* Základní algoritmus i jeho varianty lze nalézt v článku (De La Banda a kol., 2003).

Příklad. Na obrázku 3.6 je znázorněn CS-strom generovaný popsaným algoritmem pro nesplnitelnost konceptu A a ontologii $\mathcal{K}_3$ z předchozího příkladu. Nalezené MUPS odpovídají množinám $D \cup P$ pro každý „světlý" vrchol, který nemá „světlého" potomka, tedy množinám $\{1, 2\}$ a $\{1, 3\}$. Z obrázku je vidět, že systematické prohledávání nám v tomto případě ušetřilo 3 testy nesplnitelnosti.



Obr. 3.6 Příklad CS-stromu. Každý vrchol má tvar D, P. Světlé vrcholy odpovídají množinám $D \cup P$, z nichž logicky plyne nesplnitelnost konceptu *Osoba*, a tmavé vrcholy těm množinám $D \cup P$, z nichž nesplnitelnost konceptu *Osoba* neplyne. Vrcholy lemované přerušovanou čárou nebyly v průběhu algoritmu vyšetřovány.

Reiterův algoritmus

Zatímco algoritmy využívající CS-stromy systematicky prohledávaly stavový prostor všech možných množin kandidátů na MUPS a s využitím monotónosti daného logického kalkulu prořezávaly nutně splnitelné množiny, je možné k problému hledání všech MUPS přistoupit i jinak a rozdělit problém hledání všech MUPS do dvou fází:

- nalezení jednoho (libovolného) MUPS,
- nalezení všech MUPS s využitím algoritmu pro nalezení jednoho MUPS.

První z problémů je řešitelný v lineárním čase například tímto jednoduchým algoritmem: postupně přidáváme k prázdné množině axiomy z $\mathcal{K}$, dokud se A nestane nesplnitelným. Dosud vzniklou ontologii si označme jako $\mathcal{K}'$. V druhé fázi postupně odstraňujeme z $\mathcal{K}'$ jednotlivé axiomy. Axiom vrátíme do $\mathcal{K}'$ pouze, zůstane-li po jeho vyjmutí A nesplnitelný. Pro další popis označíme jako *singleMUPS*($\mathcal{K}$, A) jeden (libovolný) MUPS pro A v $\mathcal{K}$ nalezený právě popsaným jednoduchým postupem.

Využitím algoritmu *singleMUPS* můžeme nyní druhý z problémů řešit postupem známým v diagnostice pod názvem *Reiterův algoritmus*. Tento algoritmus využívá toho, že každý nalezený MUPS je minimální vzhledem k množinové inkluzi a snaží se hledat ostatní MUPS v každé ontologii vzniklé z $\mathcal{K}$ odstraněním jednoho z axiomů prvního nalezeného MUPS. Je-li takto nalezen další MUPS, postup se opakuje jako pro první nalezený MUPS. Nejlépe Reiterův algoritmus popíšeme opět pomocí stromové struktury, (hitting set tree). Vrcholy tohoto stromu jsou tvaru ($\mathcal{K}_i$, M_i), kde $\mathcal{K}_i$ je ontologie vzniklá z $\mathcal{K}$ odstraněním některých axiomů a M_i = *singleMUPS*($\mathcal{K}_i$, A), nebo M_i = "SAT", je-li A splnitelný v $\mathcal{K}_i$. Hrana vede od vrcholu ($\mathcal{K}_i$, M_i) k vrcholu ($\mathcal{K}_i \setminus \{\alpha\}$,...), pokud $\alpha \in M_i$.

1. INICIALIZACE. Nalezneme pomocí algoritmu *singleMUPS* libovolný MUPS pro A v $\mathcal{K}$. Ten bude reprezentován kořenem $s_0 = (\mathcal{K}, M)$. Dále položme $Z = (s_0)$.

2. PROHLEDÁVÁNÍ DO HLOUBKY. Je-li Z prázdný ukončíme prohledávání. Jinak vyjmi první prvek ze Z a označ jej $s_i = (\mathcal{K}_i, M_i)$.

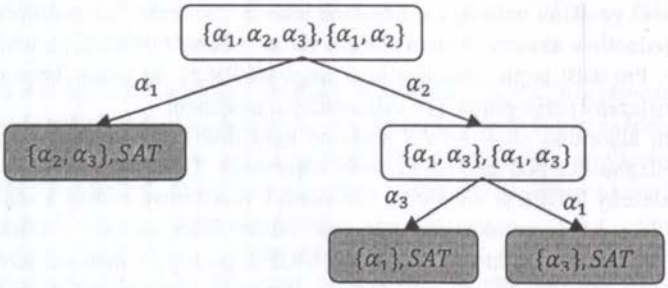
3. TEST. Je-li M_i = SAT, potom pokračujeme prohledáváním grafu v bodě 2.

4. ROZKLAD M_i. Pro každý $\alpha \in M_i$ vložíme do Z nový vrchol $(\mathcal{K}_i \setminus \{\alpha\}, singleMUPS(\mathcal{K}_i \setminus \{\alpha\}, A)$. Pokračujeme prohledáváním v bodě 2.

Při analýze korektnosti a úplnosti tohoto algoritmu je třeba si uvědomit zejména to, že MUPS nalezený v nějaké podmnožině $\mathcal{K}$ je minimální i vzhledem ke $\mathcal{K}$ a že krokem 4 pokryjeme všechny možnosti, jak mohou vypadat MUPS pro $\mathcal{K}_i$, známe-li jeden z nich (M_i).

Po skončení algoritmu jsou ve vrcholech, pro něž $M_i \neq$ SAT, jednotlivé množiny MUPS. Minimální cesty (vzhledem k inkluzi hran) potom představují příslušné *diagnózy* (z pohledu teorie množin jde o *hitting množiny* systému všech MUPS), tedy minimální množiny axiomů, jejichž současné odstranění z $\mathcal{K}$ zajistí splnitelnost vyšetřovaného konceptu A. Automatické vygenerování těchto diagnóz a jejich prezentace tvůrci ontologie je samozřejmě z uživatelského hlediska velmi užitečná. Diagnózy vlastně představují *návrhy na řešení* problému nesplnitelnosti – jsou-li všechny axiomy v diagnóze současně odstraněny z $\mathcal{K}$, stane se vyšetřovaný koncept splnitelným. A právě možnost, že získáme tyto diagnózy, představuje významnou výhodu Reiterova algoritmu oproti algoritmům založeným na CS-stromech. Ilustrujme nyní Reiterův algoritmus na následujícím příkladu:

Příklad. Uvažujme stejný problém jako v předchozí kapitole. Na obr. 3.7 je znázorněn průběh Reiterova algoritmu. Hledané MUPS $\{\alpha_1, \alpha_2\}$ a $\{\alpha_1, \alpha_3\}$ jsou určeny ohodnocením světlých vrcholů. Diagnózy jsou naproti tomu určeny minimálními cestami, tedy $\{\alpha_1\}$ nebo $\{\alpha_2, \alpha_3\}$. Cesta určující množinu axiomů $\{\alpha_1, \alpha_2\}$ není diagnózou, neboť není minimální.



Obr. 3.7 Reiterův algoritmus a stromová reprezentace hitting množin. Každý vrchol má tvar $\mathcal{K}_i$, $singleMUPS(\mathcal{K}_i, A)$. Tmavé vrcholy reprezentují $\mathcal{K}_i$, ve kterých je koncept A splnitelný.

3.4.2 Dotazování v deskripčních logikách

V odstavci 3.2.2 jsme si představili základní inferenční procedury systémů založených na deskripční logice. Praktické použití takových systémů je však podmíněno existencí rozhraní, které umožní pokládat dotazy složitější než pouhé testování konzistence ontologie či splnitelnosti konceptu. Přirozené rozšíření základních inferenčních procedur, analogické databázím, představují *konjunktivní dotazy do ABoxu*. První ucelený návrh

jazyka konjunktivních dotazů do ABoxu byl představen v práci (Horrocks a kol., 2000) pro deskripční logiku *ALC*. Bez újmy na obecnosti lze však zavedený formalismus aplikovat i na logiky expresivnější, tedy i ty, které jsou základem jazyků OWL a OWL 2.

Kromě konjunktivních dotazů do ABoxu jsou však, zejména v poslední době, studovány expresivnější dotazovací jazyky, jako např. SPARQL DL (Sirin a kol. 2007), (Křemen a kol. 2008) nebo OWL SAIQL (Kubias a kol., 2007), které vedle individuálů umožňují parametrizovat též koncepty a role. Tyto (ve své podstatě meta-) jazyky umožňují získávat informace o vzájemných vztazích konceptů a rolí a individuálů navzájem. Například jazyk SPARQL DL umožňuje formulovat dotaz tvaru *Type*(*?x*, *Student*), *Type*(*?x*, *?c*), *SubClassOf*(*?c*, *Zaměstnanec*), který nalezne pro každého studenta, který je též zaměstnancem, jeho zaměstnanecký poměr (např. *Výzkumník*, *AdministrativníPracovník*, *Manažer* aj.). Rozdíl proti standardním konjunktivním dotazům do ABoxu představeným v dalším textu je v tom, že proměnná *?c* není vázána na individuály z dané ontologie, nýbrž na pojmenované koncepty z dané ontologie. Spokojme se pouze s tímto jednoduchým příkladem, který by měl posloužit jako motivace pro další studium, neboť důkladnější analýza těchto jazyků přesahuje rámec této kapitoly.

3.4.2.1 Konjunktivní ABox dotazy

Konjunktivní ABox dotaz Q uvažujeme ve tvaru

$$Q(?x_1,\ldots,?x_D) = t_1,\ldots, t_T,$$

kde každý atom t_i je buď $C(y_k)$, nebo $R(y_k, y_l)$, kde y_k, y_l mohou být jak *individuály* z *NI* dotazované ontologie $\mathcal{K}$ se signaturou (*NC*, *NR*, *NI*), tak *proměnné* z nově zavedené množiny proměnných $V = (?x_1, ?x_2,\ldots)$ po dvou disjunktní s *NI*, *NR* a *NC*.

Je-li dotaz tvaru $Q()$, nazýváme jej *booleovský* – ten ověřuje existenci hledané struktury v každém modelu ontologie $\mathcal{K}$. Jeho sémantiku lze odvodit od interpretace $I = (\Delta^I, \bullet^I)$ pro $\mathcal{K}$ rozšířené o ohodnocení proměnných. *Ohodnocení* υ je funkce z $NI \cup V$ do Δ^I shodující se s $\bullet^I$ na funkčních hodnotách pro prvky *NI*. Řekneme, že interpretace I splňuje atom $t_i = C(y_k)$ (popř. $R(y_k, y_l)$) vzhledem k υ (píšeme $I \models_\upsilon t_i$), pokud $\upsilon(y_k) \in C^I$ (popř. $\langle \upsilon(y_k), \upsilon(y_l)\rangle \in R^I$). Dále interpretace I splňuje (tzv. booleovský) dotaz $Q() = t_1,\ldots,t_T$ (označme $I \models_\upsilon Q()$), pokud $I \models_\upsilon t_i$ pro všechna $1 \le i \le T$. Z hlediska vyhodnocování dotazu $Q()$ není konkrétní ohodnocení υ příliš zajímavé, pouze požadujeme, aby alespoň jedno existovalo: dotaz $Q()$ je splnitelný (označíme $I \models Q()$), pokud existuje υ takové, že $I \models_\upsilon Q()$. Analogicky zavedeme logický důsledek: $\mathcal{K} \models Q()$, pokud $I \models Q()$ kdykoliv $I \models \mathcal{K}$.

Z praktického hlediska jsou důležité dotazy s argumenty (nebooleovské), jejichž vyhodnocením získáme množinu n-tic individuálů. Sémantika takového dotazu je přímým rozšířením sémantiky booleovského dotazu: n-tice $\langle i_1,\ldots, i_n\rangle$ *patří do výsledku* dotazu $Q(?x_1,\ldots, ?x_n)$, pokud booleovský dotaz vzniklý z Q substitucí všech i_k za $?x_k$ je splnitelný. Proměnné vyskytující se v argumentu dotazu ($?x_k$ pro $1 \le k \le D$) nazýváme *rozlišené* (distinguished variables), ostatní proměnné ($D \le k$) jsou *nerozlišené* (undistinguished variables). Zatímco ohodnocením nerozlišené proměnné může být libovolný

doménový element, rozlišenou proměnnou lze ohodnotit pouze individuálem vyskytujícím se v (některém axiomu) $\mathcal{K}$.[8]

Příklad. Mějme jednoduchou ontologii $\mathcal{K}_4 = (\{\},\{(\exists R_1.C_1)(i), R_2(i,j), C_2(j)\})$. Dotaz $Q_1() = R_1(?x_1, ?x_2)$ je logickým důsledkem $\mathcal{K}_4$. Výsledkem dotazu $Q_2(?x_1) = R_1(?x_1, ?x_2)$ je množina $\{\langle i\rangle\}$ a výsledkem dotazu $Q_3(?x_1) = R_1(?x_1, ?x_2)$ *je prázdná množina.*

Podívejme se nyní, jakým způsobem lze ověřovat splnitelnost booleovských dotazů. Jednoduchý způsob pro vyhodnocování booleovských dotazů byl představen pro logiku *ALC* v (Horrocks a kol., 2000). Daný konjunktivní dotaz se transformuje pomocí tzv. *zabalovací procedury* (rolling-up technique) do konceptu, který popisuje jeho strukturu. Tato procedura zabaluje dotaz iterativně, kdy v každém kroku se provede jedna z těchto transformací:

- pro každý individuál y se na konec dotazu přidá atom $C_y(y)$ a do prohledávané ontologie axiom $C_y(y)$. Koncept C_y se nazývá *reprezentativní koncept* individuálu y a představuje způsob, jakým zajistit v dotazu identitu individuálu po jeho zarolování.
- nahradí se každý atom tvaru $R(y_k, y_l)$ atomem $(\exists R.\top)(y_k)$, pokud se y_l nevyskytuje v dotazu v žádném dalším atomu dotazu.
- nahradí se každý atom tvaru $R(y_k, y_l)$ atomem $(\exists R.C)(y_k)$, pokud se y_l vyskytuje v dotazu jediného dalšího atomu, a to tvaru $C(y_l)$ a
- jsou nahrazeny každé dva atomy tvaru $C_1(y_k)$ a $C_2(y_k)$ atomem $(C_1 \sqcap C_2)(y_k)$.

Po této transformaci získáme transformovaný dotaz s pravou stranou tvaru $C(x)$. Splnitelnost konjunktivního dotazu se tímto způsobem převede na buď ověření $\mathcal{K} \models C(x)$, je-li x individuálem, nebo ověření nekonzistence ontologie $\mathcal{K} \cup \{C \sqsubseteq \bot\}$, je-li x nerozlišenou proměnnou.

Příklad. Dotaz $Q_4() = R_1(?x_1, ?x_2), R_2(?x_1, ?x_3), C(?x_3)$ lze pomocí této procedury nahradit novým $Q_4'() = (\exists R_1.\top \sqcap \exists R_2.C_2)(?x_1)$. Dále $\mathcal{K}_4 \cup \{\exists R_1.\top \sqcap \exists R_2.C_2 \sqsubseteq \bot\}$ je nekonzistentní a Q_4' a tedy i $Q_4()$, je logickým důsledkem $\mathcal{K}_4$.

Jak již bylo zmíněno v předešlém textu, umožňuje logika *ALC* pomocí axiomů TBoxu axiomatizovat pouze modely ve tvaru kořenového stromu. Proto je i tato jednoduchá metoda bohužel aplikovatelná pouze na dotazy, které mají stromovou strukturu, a nelze ji použít v obecném případě pro dotazy obsahující cykly,[9] např.

$$Q_5() = R(?x_1, ?x_2), R(?x_2, i), R(i, ?x_1).$$

Tento problém lze částečně vyřešit v expresivních deskripčních logikách (např. *SHOIN*), které umožňují modelovat inverzní role I a nominály O. Pro tyto jazyky lze rozšířit uvedenou zabalovací proceduru o možnost zabalit atomy $R(y_k, y_l)$ v opačném směru, tedy

[8] Uvedené tvrzení není vzhledem k definici υ úplně korektní, neboť ohodnocení υ mapuje proměnné a individua do prvků domény, nikoliv do prvků *NI*. Jistě však v každé interpretaci musí být každý prvek *NI* interpretován jako právě jeden prvek domény – na tyto prvky uvedené tvrzení odkazuje.

[9] Uvažujeme široce přijímanou grafovou reprezentaci konjunktivních dotazů, kde každý term $R(y_k, y_l)$ odpovídá hraně mezi vrcholy y_k a y_l ohodnocené R a každý term $C(y)$ odpovídá vrcholu y ohodnocenému C.

pomocí konceptu $(\exists R^{-}.\top)$ a zabalit cykly, které obsahují alespoň jeden individuál. Například $Q_5(\,)$ by bylo možné zabalit do

$$(\exists R.(\exists R.(\exists R.\{i\})))(\{i\}).$$

Výhodou tohoto jednoduchého přístupu založeného na zabalování dotazu je možnost využití existujících tablových algoritmů pro ověřování konzistence v deskripčních logikách. Bohužel je však není možné použít pro vyhodnocování dotazů s cykly tvořenými nerozlišenými proměnnými. Problémy s takovými cykly lze pro některé podmnožiny logiky *SHOIN* vyřešit pomocí propracovanějšího tablového algoritmu představeného v sborníku (Glimm a kol., 2007). Jde o standardní verzi tablového algoritmu pro ověřování konzistence dané deskripční logiky (*SHIQ*), v níž je blokovací algoritmus změněn tak, aby blokování nenastalo příliš brzy a bylo možné celý konjunktivní dotaz vyhodnotit syntaktickým mapováním na všechny grafy zúplnění, které neobsahují přímý spor. Použití tohoto přístupu je omezeno na dotazy, které neobsahují atomy tvaru $R(y_k, y_l)$, kde R je tranzitivní, nebo existuje tranzitivní $R_2 \sqsubseteq R$. I přesto však zatím není znám algoritmus k vyhodnocení libovolných konjunktivních dotazů pro jazyk expresivity alespoň *SHOIN* (a tedy ani pro jazyk sémantického webu OWL DL, ani OWL 2 DL).

Až dosud jsme se zabývali dotazy bez rozlišených proměnných. Schopnost získávání individuálů daných vlastností je však z hlediska uživatele DL systému nezbytná. Z výpočetního hlediska je vyhodnocování dotazů s rozlišenými proměnnými jednodušší. Čtenáře jistě napadne, že stačí každou rozlišenou proměnnou nahradit postupně každým individuálem z $\mathcal{A}$, a otestovat vzniklý booleovský dotaz pomocí technik popsaných v předchozím odstavci. Tento postup je samozřejmě neefektivní a lze jej urychlit například prořezáním možných kandidátů pro každou proměnnou pomocí výše uvedeného *zabalení* dotazu. My se těmito technikami nebudeme podrobněji zabývat a zájemce odkážeme na práci (Horrocks a kol., 2000).

3.5 Deskripční logiky a pravidlové systémy

Na závěr je užitečné se zmínit o souvislostech mezi deskripčními logikami a jazykem Prolog – prominentním reprezentantem predikátového kalkulu prvního řádu. Prologovský program využívá předpoklad konečné domény (closed world assumption, CWA) a neumožňuje tak modelovat klasickou logickou negaci (máme-li nekonečnou doménu, potom pro každý unární predikát P a každou interpretaci I buď $|P^I|$ nebo $|\neg P^I|$ je nutně nekonečná). Naproti tomu deskripční logiky omezují expresivitu modelované znalosti při zachování otevřené (nekonečné) domény. Právě tato vlastnost je předurčuje pro úlohy s neúplnou, popř. distribuovanou, znalostí.

3.6 Závěrečné poznámky

Kromě jazyka *ALC* a jeho rozšíření stručně představených v této kapitole je v současné době věnována velká pozornost i dalším deskripčním logikám, které mají lepší výpočetní

vlastnosti než jazyk *ALC*, nebo OWL DL za cenu omezení vyjadřovacích schopností. Mnoho těchto jazyků je definováno jako *profily* (fragmenty) logiky *SROIQ*, která je základem standardu sémantického webu OWL 2. Jedním z nejvýznamnějších představitelů tohoto trendu je jazyk DL Lite a jeho varianty (Calvanese a kol., 2004). Ten umožňuje převést konjunktivní dotazy nad ontologiemi na databázové dotazy (např. pomocí jazyku SQL). Vyhodnocování konjunktivních dotazů se tedy stane časově polynomiální vzhledem k velikosti dat.

Literatura

Baader F., Calvanese D., McGuinness D., Nardi D., Patel-Schneider P.: *The Description Logic Handbook, Theory, Implementation and Applications*. Cambridge Press, 2003.

Borgida A., Brachman R., McGuinness D. L., Resnick L. A.: *CLASSIC: a structural data model for objects*, ACM SIGMOD, 1989.

De La Banda M. G., Stuckey P. J., Wazny J.: Finding All Minimal Unsatisfiable Subsets. In: *PPDP'03*, 2003.

Calvanese D., Lenzerini M., Rosati R., Vetere G.: DL-Lite: Practical Reasoning for Rich DLs. In: *DL 2004*, 2004.

Glimm B., Horrocks I., Lutz C., Sattler U.: Conjunctive Query Answering in the Description Logic SHIQ. In: *IJCAI*, 2007.

Horrocks I., Sattler U.: A Description Logic with Transitive and Inverse Roles and Role Hierarchies. In: *Description Logic Workshop 98*, 1998, s. 72–81.

Horrocks I., Sattler U.: Decidability of SHIQ with Complex Role Inclusion Axioms. *Artificial Intelligence,* 2004.

Horrocks I., Sattler U.: A tableaux decision procedure for SHOIQ. In: *Int. Joint Conf. on Artificial Intelligence*, 2005, s. 448–453.

Horrocks I., Tessaris S.: *Querying the Semantic Web: a Formal Approach.* In: *ISWC*, 2002, s. 177–191.

Horrocks I., Tessaris S.: A Conjunctive Query Language for Description Logic ABoxes. In: *AAAI/IAAI*, 2000, s. 399–404.

Kalyanpur A., Parsia B., Horridge M., Sirin E.: Finding All Justifications of OWL DL Entailments. In: *ISWC/ASWC*, 2007, s. 267–280.

Křemen P., Sirin E.: SPARQL-DL Implementation Experience, In: *OWLED*, 2008.

Kubias A., Schenk S., Staab S., Pan J.: OWL SAIQL – An OWL DL Query Language for Ontology Extraction, In: *OWLED*, 2007.

Mařík V., Štěpánková O., Lažanský J. a kol.: *Umělá Inteligence 1*. Academia, Praha 1993.

Schlobach S., Huang Z., Cornet R., Van Harmelen F.: *Debugging Incoherent Terminologies*. Technická zpráva Vrije Universiteit Amsterdam, 2006.

Schmidt-Schauss M., Smolka G.: Attributive concept descriptions with complements. *Artifical Intelligence*, 1, 1991, s. 1–26.

Schmolze J. G., Beranek B., Inc N.: An overview of the KL-ONE knowledge representation system, *Cognitive Science*, 1985.

Sirin E., Parsia B.: SPARQL-DL: SPARQL Query for OWL-DL, In: *OWLED*, 2007.

Zolin E.: *Description Logic Complexity Navigator*. On-line na adrese http://www.cs.man.ac.uk/~ezolin/dl/, 2007, 10. 10. 2008.

4 Ontologické inženýrství na sémantickém webu

4.1 Úvod

Jak je z názvu patrné, tato kapitola tematicky bezprostředně navazuje na kapitoly předcházející. S řadou problémů a přístupů týkajících se tvorby ontologických modelů, tj. ontologického inženýrství, se čtenář již mohl seznámit v kapitole 1. Využívání jazyka OWL a dalších modelů a nástrojů vzniklých v rámci „světa" sémantického webu, o kterém pojednávala kapitola 2, však přináší do ontologického inženýrství řadu specifických rysů.

Prvním z nich je strukturní syntax a formálně logická sémantika *jazyka OWL*. V oddílu 4.2 postupně rozebereme některá úskalí a rozhodovací situace, které musí ontologický inženýr řešit na úrovni volby prvků tohoto jazyka, jejich pojmenování a vzájemného propojování, a to od elementárních konstrukcí až po rozsáhlejší logické vzory. Problémy spojené s logickou sémantikou jazyka jako takového zde ovšem budou spíše jen nastíněny a ilustrovány na příkladech; podrobnější a rigoróznější zpracování této problematiky má čtenář k dispozici v kapitole 3 věnované deskripčním logikám jako formálnímu zázemí odvozování v OWL.

Významnou roli však v sémantickém webu hraje také otázka přímého přepoužívání (reuse) existujících modelů a jejich částí, zejména obecných ontologií a tzv. obsahových návrhových vzorů; tomuto tématu se věnuje oddíl 4.3.

Vzhledem k úloze, kterou v současnosti v ontologickém inženýrství *návrhové vzory* hrají, se pak v oddílu 4.4 nad jejich podstatou a rolí hlouběji zamyslíme a rozebereme jejich aktuálně navrhovanou typologii, zejména s odvoláním na „standardizační" portál *ontologydesignpatterns.org*. Uvedeme si konkrétní příklad obsahového návrhového vzoru implementovaného v OWL a zmíníme se o metodice, která přepoužívání takových vzorů podporuje.

Závěrečný oddíl 4.5 shrnuje hlavní zásady ontologického inženýrství v kontextu sémantického webu (i s ohledem na nově nastupující technologii Linked Data) a odkazuje na další rozšiřující literaturu.

V kontextu celého svazku (popř. knižní série) je na místě charakterizovat vztah ontologického inženýrství a oboru *umělé inteligence*. Lze říci, že ontologické inženýrství na oboru umělé inteligence participuje nepřímo, ovšem správnost, úplnost a celková „použitelnost" ontologických modelů má zásadní vliv na „inteligentní" chování softwarových aplikací, které tyto modely využívají. Zároveň ale ontologické inženýrství oblast umělé inteligence v jistém směru přesahuje. Důležitým hlediskem tvorby a využívání ontologií je totiž zachycení konceptuálních znalostí člověka (jako produktu „pojmotvorné" aktivity přirozeného intelektu) a jejich sdílení mezi lidmi navzájem.

Kapitola se soustřeďuje na ontologické inženýrství „v malém". Neklade si tedy za cíl formulovat postupy pro tvorbu velmi rozsáhlých ontologických modelů, kdy se navíc

uplatňují principy týmové práce, vzory architektur nebo propojování ontologií s „neontologickými" (např. objektovými) modely. Takové postupy jsou ve své většině nezávislé na použitém ontologickém jazyku. Tématu se lehce dotýká kapitola 6 věnovaná aplikacím ontologií ve specifické věcné doméně – průmyslové výrobě.

4.2 „Vnitřní rozměr" ontologického inženýrství – struktura a sémantika jazyka

4.2.1 Volba typů entit v jazyce OWL

Přestože není inventář typů entit v OWL a RDFS[1] příliš rozsáhlý, výběr vhodného typu (individua, třídy, objektové či datové vlastnosti) pro určitý pojem v určitém modelu není vždy triviální. Základním principem je přiřadit správnou množinovou interpretaci[2] dané entity vzhledem k „univerzu" uvažovaných objektů:

- Interpretací *individua* je objekt z univerza;
- Interpretací *třídy* je množina objektů z univerza;
- Interpretací *objektové vlastnosti*[3] je množina uspořádaných dvojic objektů z univerza;
- Interpretací *datové vlastnosti* je množina uspořádaných dvojic, kde prvním členem je objekt z univerza a druhým literál, tj. hodnota datového typu (řetězec, číslo, datum apod.).

Tímto se ovšem problém do jisté míry jen přesouvá na rozhodování, zda daný pojem odpovídá jednotlivému objektu, množině objektů či množině uspořádaných dvojic. Záleží na míře granularity dané konceptualizace, která je, jak už bylo ukázáno v kapitole 1, pro tvůrce ontologie do jisté míry volitelná.

Příklady s více možnostmi volby

Notorickým příkladem jsou biologické taxony, které můžeme chápat jako již nedělitelná individua nebo jako třídy. Entita „kočka domácí"[4] může být chápána jako instance třídy „Živočišný druh" a množinově interpretována jako individuum, může ale být také chápána jako třída a množinově interpretována jako množina biologických organismů. Obdobná situace může nastat v případě vztahů (či událostí zahrnujících více než jednoho účastníka), kde se pro změnu rozhodujeme mezi třídou a objektovou vlastností (pro konkrétní případy vztahů a událostí pak samozřejmě mezi instancemi takové třídy, resp. vlastnostmi). Například vztah mezi kupujícím a výrobkem můžeme modelovat pomocí třídy „Nákup" nebo vlastnosti „nakoupil"; událost spočívající ve srážce dvou aut můžeme modelovat třídou „Srážka aut" nebo vlastností „srazil se s". V případě objektů mapovatelných na jednoduchá data můžeme mít navíc volbu mezi třídou a datovou vlastností.

---

[1] Základní přehled obou jazyků byl uveden v kap. 2.

[2] Pojem „interpretace" zde používáme v intuitivním a ne zcela přesném pojetí. Formální definice interpretace symbolů v ontologickém modelu je podána v kap. 3.

[3] Připomínáme, že pojem vlastnosti je v oblasti sémantického webu víceméně z historických důvodů používán k označení binární relace.

[4] V tomto odstavci pro lepší čitelnost používáme názvy entit s diakritikou a mezerami. Pokud by se jednalo o identifikátory – URI v reálných implementovaných ontologiích – vedlo by používání diakritiky a mezer ke komplikacím. Na druhou stranu může mít každá entita v OWL přiřazené hodnoty anotační vlastnosti *rdfs:label*, které lze v aplikacích používat jako jména entit, viz odst. 4.2.2.

Například vztah mezi člověkem a jeho jménem může mít charakter datové vlastnosti, pokud nás zajímá pouze jméno jako textový řetězec. Na druhé straně můžeme jméno chápat jako abstraktní entitu (modelovanou jako třídu), jejíž instance teprve následně odkazují například na křestní jméno, příjmení, v některých případech navíc na prostřední iniciálu, nebo může mít (v neevropských jazycích) i výrazně odlišnou strukturu. V takovém případě musíme ovšem namísto datové vlastnosti do ontologie zařadit objektovou vlastnost, která bude v oboru hodnot mít příslušnou třídu.

Kritéria pro volbu typu

Z hlediska možnosti dále ontologii obohacovat, a také z hlediska využitelnosti odvozovacích nástrojů, je obvyklou první volbou využití tříd. V jazyce OWL ve verzi podporované aparátem deskripční logiky se nepředpokládá, že bychom instanci dodatečně začali chápat jako třídu[5] a přiřazovat jí její vlastní instance (dejme tomu, kdyby měl informační systém zoologické zahrady vedle chovaných druhů začít evidovat i jednotlivé exemplář). Vzhledem k omezení vlastností v OWL na binární (relace) není ani možné přidat dodatečný argument vlastnosti (např. pokud bychom dodatečně chtěli vedle kupujícího a výrobku zachytit také prodávajícího nebo vedle identity obou aut zaznamenat také místo a čas srážky).

Na druhou stranu extrémní míra vyjadřování typicky „relačních" objektů pomocí tříd (vyjádření relace pomocí třídy označujeme jako tzv. reifikaci, kterou však nelze zcela ztotožnit s reifikací jako syntaktického konstruktu jazyka RDF, viz kap. 2) může vést k nadměrnému rozsahu a složitosti ontologie. Při preferenci tříd oproti individuím je zase třeba dávat pozor, aby zamýšlená množinová interpretace určité třídy nebyla ve skutečnosti specifickým vztahem mezi entitami různého typu; k tomu by došlo například, pokud bychom divadelní hru R.U.R modelovali jako třídu a jako její instance chápali její jednotlivá představení, nebo kdybychom jako třídu chápali určitý algoritmus a jako jeho instance jednotlivé běhy algoritmu, popřípadě kdybychom dokonce firmu chápali jako třídu obsahující jako instance své zaměstnance. Ve všech těchto případech je nutné buďto použít individuum, např. „R.U.R.", a spojit ho s individuálními představeními objektovou vlastností, která se může jmenovat například „bylo provedeno", nebo, pokud chceme primárně „množinově" zpracovávat tématiku představení, zavést třídy typu „Představení R.U.R" obsahující patřičné instance.

Problém volby mezi třídou a individuem nemusí být ovšem jednoznačný ani v případě, kdy si pro třídu dovedeme její instance představit zcela korektním způsobem. Představme si, že bychom chtěli entitě „Kočka domácí" ze zoologické ontologie přiřadit (např. pomocí relace „pojednává o") jednak knihy obsahující sbírky příběhů a fotografických snímků o konkrétních kočkách a jednak odbornou knihu, která pojednává o chovu koček z teoretického hlediska, včetně rozlišení specifických ras. Zatímco v prvním případě bychom mohli pro danou třídu vytvořit instance (v případě, že identita konkrétních koček není pro aplikaci zajímavá, může jít o prázdné uzly v reprezentaci RDF), ke kterým by se kniha vztahovala, ve druhém případě kniha skutečně pojednává také o pojmu „Kočka domácí" jako takovém, včetně jeho množinové sémantiky. Některé

[5] Pro přesnost uveďme, že nová verze standardu, OWL 2, umožňuje použít pro dvě odlišné entity, např. třídu a instanci, stejný identifikátor (URI). Taková konstrukce, označovaná jako *punning* („pun" je anglický výraz pro slovní hříčku využívající záměnu různých významů určitého slova), ovšem nemá vliv na logické odvozování, v rámci kterého entity zůstávají oddělené.

přístupy proto navrhují v takovém případě zavést individuum typu „Kočka domácí", protože v sémantice deskripční logiky lze pomocí relace spojovat pouze individua. K problému se ještě vrátíme v odst. 4.2.4 věnovaném logickým návrhovým vzorům.

Poznamenejme ještě na závěr, že v ontologiích jako znalostních „schématech" zpravidla nechceme zachycovat jednotlivé objekty. To je úkol znalostních bází, které jsou na těchto ontologiích postavené; z formálně logického hlediska je ovšem přirozené chápat ontologii a znalostní bázi jako jeden celek – množinu formulí deskripční logiky zahrnující jak tzv. TBox (tvrzení o třídách a vlastnostech), tak i ABox (tvrzení o individuích). Individua patří do ontologie tehdy, pokud je třeba s jejich pomocí definovat některou třídu, popřípadě pokud jde o unikátní objekty zásadního významu pro doménu: kosmické objekty, geopolitické a geografické útvary (státy, kontinenty), chemické prvky apod. Specifickým typem individuí pak mohou být vyjmenované hodnoty veličin, např. hodnota „Vysoký krevní tlak" pro veličinu „Krevní tlak" v medicínské ontologii.

4.2.2 Jmenné konvence pro názvy entit

Přestože ústřední složkou ontologií v OWL je jejich logická struktura, nelze z řady důvodů zanedbat jejich provázanost s termíny a strukturou přirozeného jazyka. Na sémantickém webu, ještě více než v tradičních (centralizovaných) znalostních aplikacích, je pro správné využití znalostních modelů důležité jejich důkladné pochopení ze strany lidí, kteří často s původními tvůrci modelů nemají možnost přímo komunikovat. Formální složka jazyka obsahuje pro tento účel příliš málo informací.

Lexikální informace v ontologiích

Dodatečné informace lze zařadit buďto přímo do identifikátorů (URI) entit, nebo do anotací.

V prvním případě je lexikální informace úzce spojena s logickou identitou entit, což je někdy předmětem kritiky. V obavě, že uživatel příliš ztotožní textový název entity s její věcnou podstatou a bude ignorovat její formální definice pomocí axiomů, někteří návrháři dokonce úmyslně používají kryptická URI typu alfanumerického kódu. Poznamenejme, že změna jména entity na úrovni URI je vlastně ekvivalentní (lokálnímu) zrušení jedné entity a současnému zavedení jiné – proto ontologické editory občas zařazují, z uživatelského hlediska poněkud překvapivě, přejmenování entity do operací typu „refaktoring ontologie".

Ve druhém případě jde o uložení lexikální informace do hodnoty anotačních vlastností *rdfs:label* (pro „lidsky srozumitelná" jména entit) a *rdfs:comment* (pro podrobnější vysvětlení), i když nový standard OWL 2 (Motik, Patel-Schneider, Parsia, 2009) otevírá pro využívání anotací daleko širší prostor. Při využívání anotací nehrozí kolize s URI využívanými v existujících zdrojích; nevýhodou ovšem je, že textové údaje tam jsou uloženy poněkud odděleně od logické části ontologie a nelze spoléhat na to, že všechny uživatelské nástroje budou (např. při výpisech axiomů, viz příklad v dalším textu) natolik „inteligentní", aby je tam „dohledaly".

V případě tvorby nových ontologií je podle většinového názoru vhodné dodržovat určité jmenné konvence již při tvorbě URI, a do anotací umístit doplňující informace, zejména jazykové varianty a slovní definice pojmů. Úprava jmen entit na úrovni URI v existujících (zejména široce využívaných) ontologiích je již samozřejmě problematičtější, neboť vyžaduje zavedení alternativních URI (aliasů) navzájem spojených logickou

konstrukcí vyjadřující ekvivalenci (v OWL 2 jde o axiom typu *EquivalentClasses*, *EquivalentObjectProperties* nebo *EquivalentDataProperties*), s možností označit původní entitu s nevhodným názvem jako zastaralou (anotační vlastností *owl:deprecated*).

Poznamenejme, že existují i „důkladnější" přístupy, které k ontologii vytvářejí samostatnou lexikální komponentu s vlastní strukturou pokrývající více jazyků, viz (Buitelaar, 2009), (Espinoza, 2009). Takové řešení je vhodné spíše pro rozsáhlé ontologie.

Populární jmenné konvence

O jmenných konvencích se vyskytuje stručná zmínka v materiálech pro začínající návrháře ontologií (Allemang, Hendler, 2011), (Noy, McGuinness, 2001), podrobněji se tématu věnují články (Schober et al., 2009), (Svátek, 2009). Publikované práce uvažují výhradně konvence pro angličtinu jako jazyk drtivé většiny formálních ontologií, proto se na ně omezíme i zde.

Uveďme si několik jmenných konvencí s různým stupněm rozšířenosti:

1. Jména tříd by měla být v *jednotném čísle* (v souladu s transkripcí vztahu instance a třídy do strukturované angličtiny jako „is-a", např. „John is a Person", a ne „John is a Persons"); tato konvence je uznávána v ontologickém inženýrství všeobecně, i když i zde existují názory proti tomu.

2. Jména by měla být nezávislá na *kontextu*. Konkrétně, srozumitelnost jména by neměla záviset na tom, zda uživatel současně vidí sousední ontologické struktury. I tato konvence je obecně uznávána, ale v praxi není často nedodržována, protože zahrnutí kontextu vede k dlouhým názvům.

3. *Objektové vlastnosti* by měly být pojmenovány *slovesnou vazbou* evokující směr vazby, a nikoliv podstatným jménem. Tato konvence má v rámci komunity OWL řadu zastánců, ale často není dodržována ani zkušenými znalostními inženýry.

4. Jména tříd by měla začínat *velkým písmenem* a jména vlastností *malým písmenem*; v obou případech ale platí, že pokud jsou jména složena z více slov, každé následující slovo by mělo začínat velkým písmenem navazujícím bez mezery (tzv. „camel case", neboli řetězce připomínající tvarem velblouda, známé i z jiných oblastí informatiky). Názvem třídy tedy může být například „MarriedMan" a názvem objektové vlastnosti „marriedTo". Tato konvence je rozšířená v rámci sémantického webu; v jiných komunitách platí konvence odlišné, např. v oblasti biomedicíny je zvykem používat jako oddělovač slov ve jménu mezeru nebo podtržítko (Schober et al., 2009).

Příklad uplatnění jmenných konvencí v axiomech

Jmenné konvence mají význam pro orientaci v ontologii i v taxonomickém pohledu, který je pro většinu ontologických editorů a prohlížečů výchozí. Jejich role je však zvláště důležitá při práci s logickými axiomy. Ty jsou totiž, v případě skládání většího počtu konceptových výrazů, často obtížně srozumitelné. Jmenné konvence v takovém případě stojí po boku manchesterské syntaxe[6], uvedené v kap. 2, která byla pro tento účel specificky vyvinuta. Analogie s přirozeným jazykem, kterou manchesterská syntax zajišťuje na úrovni konstrukcí jazyka OWL samotného, zajišťují jmenné konvence na úrovni konkrétního ontologického modelu v tomto jazyce vyjádřeného.

Uveďme si jako příklad následující axiom (upraveno podle práce (Svátek, 2009))

[6] http://www.w3.org/TR/owl2-manchester-syntax/

Class: StateOwned SubClassOf: Director only nomination some ministries

a pro porovnání jeho variantu v případě, že při návrhu ontologie byly respektovány zmíněné návrhové jmenné konvence

Class: StateOwnedCompany SubClassOf: hasDirector only nominatedBy some Ministry.

Díky kombinaci klíčových slov manchesterské syntaxe (Class, SubClassOf, only, some) a jmen entit z ontologie samotné lze z axiomu ve druhé variantě poměrně snadno vyčíst sdělení, že „firma vlastněná státem může mít pouze takového ředitele, který byl jmenován nějakým ministerstvem". Konkrétní uplatnění jmenných konvencí je asi čtenáři také zřejmé:

- Vhodné jméno pro třídu firem vlastněných státem je *StateOwnedCompany* a nikoliv jen *StateOwned* (se spoléháním na přítomnost nadtřídy *Company* jako taxonomického kontextu) – uplatnění konvence 2.
- Vhodné jméno pro objektové vlastnosti označující vztah mezi organizací a jejím ředitelem je *hasDirector* a nikoliv *Director* – uplatnění konvencí 3 a 4.
- Obdobné je to pro vztah mezi osobou a institucí, která tuto osobu nominovala do nějaké funkce: *nominatedBy* vs. *nomination* (poznamenejme, že zde je model nereálně zjednodušen, protože ve vztazích chybí vazba na pozici, kam byl dotyčný nominován – zde by bylo nutné použít návrhový vzor pro *n*-ární relace, viz oddíl 4.2.4) – uplatnění konvence 3.
- Vhodné jméno pro třídu ministerstev je Ministry a nikoliv ministries – uplatnění konvencí 1 a 4.

Na závěr poznamenejme, že jmenné konvence a obecně systematický způsob pojmenovávání entit může v budoucnu nacházet stále větší uplatnění nejen pro zlepšení srozumitelnosti pro lidské uživatele, ale i pro *strojovou analýzu* ontologií, např. za účelem hodnocení jejich kvality, usnadnění mapování na jiné ontologie nebo transformace do odlišné struktury. Pro automatickou analýzu má smysl se zabývat zejména rozložením „jmenných vzorů" po struktuře ontologie, tj. nikoliv jen pro jednotlivé entity izolovaně. O takových vzorech se ještě zmíníme i v následujících dvou odstavcích.

4.2.3 Elementární struktury a inferenční efekty

Popis elementárních struktur jazyka OWL je k dispozici, včetně příkladů, v oficiálních specifikacích jazyka (Motik, Patel-Schneider, Parsia, 2009), ale například i v tutoriálu návrhového prostředí Protégé v jazyce OWL (Horridge, 2011). Cílem tohoto stručného oddílu není suplovat tyto systematické zdroje, ale spíše čtenáře, který je již se základní literaturou obeznámen a má za sebou úvodní pokusy s tvorbou ontologií, upozornit na některé méně samozřejmé nebo často porušované principy modelování, zejména s ohledem na inferenční (odvozovací) sémantiku OWL.

Množinová sémantika taxonomie tříd

Přestože s požadavkem, že všechny instance dané třídy by měly být současně instancemi každé z jejích nadtříd, každý začínající ontologický inženýr ochotně vysloví souhlas, je skutečností, že v málokteré ze začátečnických (a leckdy dokonce i profesionálních) ontologií je požadavek bezezbytku splněn. Často je za vztah *třídy a podtřídy* vydáváno to, co

je ve skutečnosti vztahem *třídy a instance* (třeba „Firma–IBM") nebo vztahem *celku a části* (třeba „Počítač–Procesor"). Někdy je také vztah samotný zamýšlen dobře, ale není vhodně realizován na úrovni pojmenování tříd, protože předpokládá interpretaci názvu třídy v kontextu nadřazených tříd (třeba Počítač–Stolní; viz jmenná konvence č. 2 v odst. 4.2.2).

Předpoklad otevřeného světa a možnost alternativních identifikátorů

Při práci s webovými ontologiemi je důležité uvědomovat si, že jsou určeny pro webové prostředí, které je rozsáhlé, heterogenní a dynamické. Jazyk OWL je proto postaven na tzv. *předpokladu otevřeného světa* („open world assumption", OWA), na rozdíl od databází nebo třeba systémů logického programování zde nepřítomnost určitého faktu v bázi znalostí neznamená, že toto tvrzení neplatí. Nemůžeme totiž vědět, jestli tato nepřítomnost není pouze důsledkem neúplnosti dané báze znalostí.

Druhou významnou odlišností oproti uzavřeným prostředím typu databází je, že nelze zaručit, že určitá entita bude vždy známá pod stejným identifikátorem. Může se stát, že jednu a tutéž entitu označí tvůrci různých bází znalostí odlišně. Mluvíme o *nepravdivosti předpokladu jedinečných jmen* („non – unique-name-assumption", non-UNA).

Pokud tedy například máme ve „webové" bázi znalostí dvě tvrzení typu, „John hasSon Peter" a „John hasSon Paul", nemůžeme si být jisti tím, že John má právě dva syny – může jich mít *víc* (jen o nich nevíme), nebo naopak může mít jen *jednoho*, který se ovšem v bázi vyskytuje pod dvěma různými jmény (resp. identifikátory URI).

Existenční a univerzální kvantifikátor

Pro informatiky zvyklé na „databázové vidění světa" může tvorba ontologie v OWL DL skrývat další úskalí, která propagátoři ontologií z iniciativy CO-ODE[7] nápaditě zviditelnili pomocí demonstrační ontologie „pizzas" a vysvětlujících materiálů k ní.

Prvním problémem je chápání existenčního a univerzálního kvantifikátoru, $\forall$ a $\exists$, v manchesterské syntaxi OWL „only" a „some". Poměrně nasnadě je asi nesprávnost definice vegetariánské pizzy jako pizzy s vegetariánskou náplní[8] (kde třída *Vegetariánská_náplň* zahrnuje podtřídy jako *Rajčatová_náplň*, *Houbová_náplň* atd.), tj. jako „Pizza *and* má_náplň *some* Vegetariánská_náplň". Pizza totiž (pokud není relace *má_náplň* deklarována jako funkce) může mít náplní víc, definice ovšem pouze požaduje, aby byla vegetariánská aspoň jedna z nich. Někoho ale možná zaskočí, že problém nevyřeší prostá výměna existenčního kvantifikátoru za univerzální, čili změna definice na „Pizza *and* má_náplň *only* Vegetariánská_náplň". Pojem vegetariánské pizzy totiž pak zahrnuje kromě opravdových vegetariánských pizz také pizzy, které nemají *žádnou* náplň (a tudíž lze říci, že všechny náplně, které mají, jsou vegetariánské). K dosažení žádoucího stavu je tedy nutné na pravé straně definice uvést konjunkci existenčního a univerzálního axiomu: „Pizza *and* má_náplň *some* Vegetariánská_náplň *and* má_náplň *only* Vegetariánská_náplň".

[7] http://www.co-ode.org/

[8] Nepříliš výstižný avšak v praxi používaný překlad anglického „topping"; kvůli němu jsou některé uvedené příklady pro českého čtenáře drobně modifikovány.

Definiční obor a obor hodnot jako odvozovací pravidla
Jinou záludností například je, že množinu všech deklarovaných podtříd určité třídy nelze apriorně chápat jako rozklad této třídy, ani tyto podtřídy nejsou vzájemně disjunktní. To může vést k problémům například ve spojení s další, pro tvůrce databází nezvyklou, skutečností, že omezení na definiční obor a obor hodnot relace se chápou jako logické formule (nad kterými probíhá odvozování), ale ne jako integritní omezení. Pokud tedy například deklarujeme, že definiční obor vlastnosti *má_náplň* je třída *Pizza* a poté uvedeme třeba u třídy *Šlehačkový_dort* (která nebude explicitně nebo odvozením prohlášena za disjunktní s třídou *Pizza*) podmínku *má_náplň some Šlehačka*, důsledkem bude, že odvozovací nástroj zařadí *Šlehačkový_dort* jako podtřídu třídy *Pizza*.

Mnoho dalších „paradoxů" je podrobně rozebráno v práci (Rector et al., 2004).

4.2.4 Rozsáhlejší logicko-strukturní návrhové vzory

Třetí skupinu problémů spojených s tvorbou ontologie ve formálním jazyce lze charakterizovat tak, že není na první pohled patrné jednoznačné převedení intuitivní konceptualizace do daného, omezeného jazyka. Pro řešení těchto problémů jsou postupně vytvářeny návrhové vzory v mnoha směrech blízké vzorům známým v softwarovém inženýrství. Ve srovnání se strukturami z předchozí kapitoly u takových vzorů zpravidla nesledujeme bezprostřední logické důsledky odvozování nad nimi, ale spíše jen jejich všeobecný vliv na možnosti odvozování nad ontologií.

***n*-ární relace**
Velmi frekventovaným problémem v OWL je nemožnost přímo modelovat relace s *aritou* větší než 2. Návrhový vzor (Noy, Rector, 2004) doporučuje pro většinu případů provést reifikaci relace, tj. *n*-ární relace (např. vztah mezi prodávajícím, kupujícím, kupovaným předmětem, popřípadě cenou a dalšími účastníky relace) je převedena na třídu (např. „Obchodní transakce") a ostatní objekty jsou již pak k jejím instancím připojeny binárními relacemi.

Třída jako hodnota vlastnosti
Složitějším problémem, řešeným vzorem s mnoha alternativami (Noy, Uschold, Welty, 2005), je situace, kdy potřebujeme na jedné straně modelovat hierarchickou strukturu (např. živočišných druhů, rodů, čeledí atd.) a na druhé straně na prvky hierarchie odkazovat relací (např. spojující určité informační zdroje se skupinami živočichů, o kterých pojednávají). Zdá se, že bychom tedy potřebovali, aby mohla samotná *třída* (např. „Kočka_domácí", viz odst. 4.2.1), a nikoliv jen její konkrétní instance (konkrétní kočky), být *argumentem relace*. To sice lze, ale pouze v dialektu OWL Full, nad kterým se odvozuje obtížněji než nad OWL DL, navíc tento postup není za všech okolností konceptuálně čistý. Jednou z alternativ je kupříkladu vytvořit k hierarchii tříd paralelní hierarchii individuí vyjadřujících různé *tematiky* (např. „tematiku koček"); relace „pojednává o" pak může odkazovat na ně a nikoliv na třídy interpretované jako množiny fyzických živočichů.

Modelování rolí
Posledním problémem (a návrhovým vzorem), o kterém se zmíníme, je potřeba odlišného modelování proměnlivých aspektů objektů oproti trvalým. Příkladem je rozlišit tvrze-

ní, že určitý objekt je člověkem, od tvrzení, že týž objekt je učitelem. Zatímco člověkem je objekt po celou dobu své existence (jde o esenciální charakteristiku ve smyslu metodiky OntoClean popsané v kapitole 1), učitelem přestane být nejen tehdy, když zemře, ale i tehdy, když změní povolání, nebo odejde do trvalého důchodu – z hlediska ontologie bychom pak museli tentýž objekt přemisťovat mezi třídami. Být učitelem je totiž *role*, kterou daný objekt hraje, ale nikoliv tzv. přirozená třída. Primitivním řešením problému je třídu *Učitel* vůbec nezavádět a spokojit se s relací *být_učitelem*. To však vede k omezenějším možnostem odvozování. Sunagawa ve své práci (Sunagawa, 2006) proto navrhuje modelovat role (např. při konverzi z některého jazyka, který má pojem role přímo ve svém metamodelu) v OWL pomocí komplexního logického vzoru. Vzor rozlišuje třídy odpovídající základním konceptům, jako je *Člověk* nebo *Škola*, třídy odpovídající rolím („role concepts"), jako je *Role_učitele*, a třídy odpovídající držitelům role („role holder"), jako je *Učitel*.

Normalizace taxonomie

V oborech s převládajícím taxonomickým stylem modelování, mezi které patří zejména biomedicína, mohou vznikat ontologie obsahující „propletené" hierarchie, kde má jedna třída větší počet nadtříd (každou odpovídající určitému hledisku klasifikace). My si tento problém pro jednoduchost přiblížíme na příkladu ontologie potravinářského zboží. Třída „Mléko_s_jahodovou_příchutí_v_plastu" by tam mohla být zahrnuta do nadtřídy „Mléčný_výrobek", ale i do nadtříd „Výrobek_v_plastovém_obalu" a „Výrobek_s_příchutí_jahody". Pro přehlednost ontologie je ale vhodnější, aby měla každá třída jen jednu nadtřídu, a to takovou, která vymezuje její přirozenou podstatu. Tou by byla třída „Mléčný_výrobek", zatímco příslušnost k ostatním nadtřídám je možné odvozovat druhotně, prostřednictvím existenčních omezení, jako je

Class: Mléko_s_jahodovou_příchutí_v_plastu *SubClassOf:* máMateriálObalu *some* Plast

ve spojení s definicí druhotné nadtřídy

Class: Výrobek_v_plastovém_obalu *EquivalentTo:* máMateriálObalu *some* Plast.

Podrobnější vysvětlení vzoru lze nalézt například v rámci katalogu ontologických vzorů Univerzity v Manchesteru.[9]

V rámci vývoje takového jazyka, jako je OWL, existuje přirozený trend k přesouvání frekventovaných konstrukcí z úrovně logických návrhových vzorů na úroveň rozšíření normy jazyka o potřebné konstrukce. Tak se například problém tzv. *kvalifikovaných kardinalitních omezení* (současný požadavek na kardinalitu relace a na třídy, jejíž instance smějí být hodnotami relace), původně nedokonale řešený pouze na úrovni návrhového vzoru, definitivně vyřešil zařazením potřebné konstrukce do nové verze jazyka, OWL 2. Podobně lze na zmíněný problém odvolání se na třídu jako na hodnoty vlastnosti aplikovat syntaktickou techniku OWL 2 označovanou jako punning (viz poznámka v odst. 4.2.1) – třída a odkazovaná instance sdílejí stejné URI (takže není třeba zavádět „novou" instanci), avšak z logického hlediska jde o dvě samostatné entity.

[9] http://www.gong.manchester.ac.uk/odp/

4.3 „Vnější rozměr" ontologického inženýrství – přepoužití externích zdrojů

Návrh ontologií je náročnou činností, která vyžaduje jak rozsáhlé znalosti v oblasti modelování, tak i porozumění dané předmětné doméně, která je modelem zachycována. Pro usnadnění procesu modelování na *obsahové rovině* je možné využít rozsáhlý inventář podpůrných prostředků. Významná je zejména specializovaná metodika *OntoClean*, jejímž cílem je omezit výskyt nejběžnějších modelovacích chyb a zajistit konzistenci výsledné ontologie – s tou jsme se již seznámili v kapitole 1. OntoClean je založen na značkování (de facto metamodelování) entit stávající ontologie vedoucí k doporučením pro její transformaci do konceptuálně správnější podoby. Další možností je přímé začlenění existujících ontologických modelů do nově vznikající ontologie. Tato možnost se používá ve dvou variacích, které se liší spíše komunitou, jež je používá, než věcnou podstatou. V prvním případě jde o přepoužití jednoduchých *slovníků*, rozšířené v oblasti Linked Data, ve druhém případě o bohatěji strukturované *obsahové návrhové vzory*.

4.3.1 Jednoduché ontologie a slovníky

V kapitole 2 byla řeč o „malých ontologiích sémantického webu". Tyto ontologie, jako je FOAF, vCard nebo SIOC, nemusí být pro definování sémantiky dat použity vždy jen přímo, ale také prostřednictvím dalších ontologií, které jejich prvky začlení do své struktury a hierarchicky pod ně „navěsí" entity speciálnější. Tak může například ontologie univerzity přiřadit třídě *foaf:Person* podtřídy „Učitel" a „Student" apod., zatímco ontologie podniku téže třídě přiřadí podtřídy typu „Zákazník" nebo „Obchodní zástupce". Přepoužití může proběhnout dvěma způsoby. Buďto jsou v kódu ontologie použity jen jednotlivé entity (tj. jejich URI je použito v axiomech nové ontologie), nebo je do cílové ontologie naimportována „malá ontologie" jako celek.

I v druhém případě se jednotlivé entity z ní používají na sobě nezávisle. Tím se tento scénář „přepoužití použití externích zdrojů" liší od následujícího, založeného na obsahových návrhových vzorech – ontologických fragmentech, které se přepoužívají jako jeden celek a zásadním způsobem tím ovlivňují celé pojetí nově vznikající ontologie.

4.3.2 Obsahové návrhové vzory

Obsahové návrhové vzory – Conceptual (nebo také Content) Ontology Design Patterns – jsou nástrojem neboli šablonou pro zjednodušení a standardizaci řešení modelovacích problémů při návrhu ontologií. Gangemi ve své práci (Gangemi, 2005) definuje některé charakteristické vlastnosti obsahových návrhových vzorů:

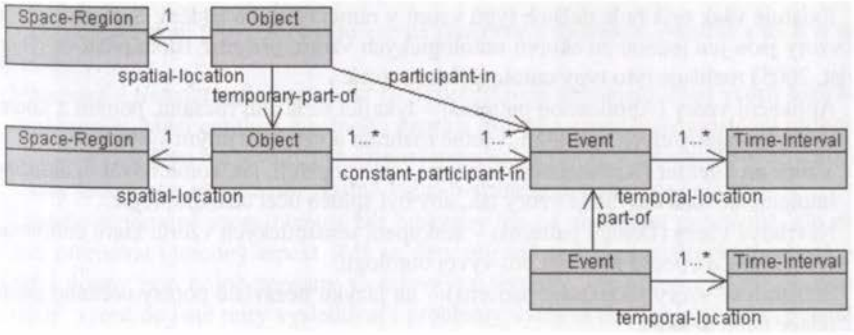
- Návrhový vzor je obvykle vytvořen na základě určité referenční základní ontologie, ale je aplikovatelný i v jiných ontologiích postavených na odlišných základních ontologiích.
- Návrhové vzory jsou obvykle dobře vizualizovatelné a jsou přiměřeného rozsahu, aby byly snadno zapamatovatelné a lehce využitelné. Návrhové vzory mohou využívat jeden druhý nebo mohou vytvářet hierarchii podle úrovně zobecnění.
- Konkretizované návrhové vzory pro danou doménu mohou být vytvořeny doménovými experty s použitím obecnějších návrhových vzorů a obecných doménových schémat.

- Návrhové vzory mohou také zachycovat doporučované postupy („best practices") pro určité oblasti modelování. Návrhové vzory mohou připomínat databázové schéma, ovšem na rozdíl od nich jsou definovány s ohledem na referenční ontologii a mají obvykle obecnější charakter.

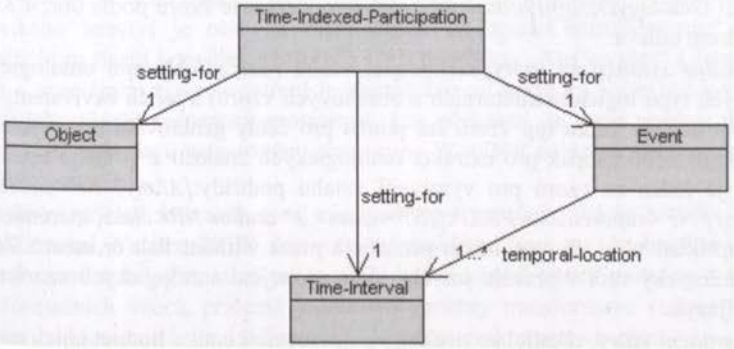
Jako nejběžnější příklady obsahových návrhových vzorů uvádí Gangemi modelování přítomnosti v časoprostorové lokaci a reifikaci časově indexované relace.

Jádro prvního vzoru, který je součástí knihovny základní ontologie DOLCE (Masolo et al., 2002), tvoří *relace účasti* (participace), která platí mezi objektem a událostí. Časová indexace je zajištěna lokalizací dané události v určitém časovém intervalu. Vzor poskytuje možnost hierarchizace objektů i událostí, jež umožňuje podporu jednoduchého odvozování. Pokud je objekt účasten události, účastní se události i jeho části. Když je objekt trvale účasten nějaké nadřazené události, účastní se i událostí, které jsou časovými částmi této nadřazené události, viz obr. 4.1.

Obsahový vzor *reifikace časově indexovaných relací* (reifikace – zvěcnění, z lat., „res" – věc, „facere" – učinit) představuje určitou alternativu pro první příklad. Abstraktní koncept *Time-Indexed-Participation* zvěcňuje vztah mezi objektem, událostí a časovým intervalem, tj. vytváří pro něj specifické entity, které ho reprezentují, viz obr. 4.2.



Obr. 4.1 Přítomnost v časoprostorové lokaci.



Obr. 4.2 Reifikace časově indexovaných relací.

Pro danou konkrétní situaci můžeme tedy často vybrat z více aplikovatelných návrhových vzorů ten nejvhodnější. Pro vytváření knihoven návrhových vzorů je doporučena popisová šablona umožňující zachytit podstatné charakteristiky každého návrhového vzoru (Gangemi, 2005).

Přestože původní pojetí obsahových vzorů abstrahovalo od konkrétního reprezentačního jazyka, z hlediska praktické využitelnosti je vhodné, aby byly vzory vyjádřeny v konkrétním jazyce, kterým je standardně OWL. Hranice mezi obsahovými a logickými vzory se v takové reprezentaci do jisté míry stírá; odlišujícím faktorem je jen větší obecnost obsahových vzorů. Příklad reprezentace obsahového vzoru v jazyce OWL v rámci portálu *ontologydesignpatterns.org* bude uveden v oddílu 4.4.

4.4 Využívání ontologických vzorů jako systematická disciplína

4.4.1 Klasifikace vzorů

Návrhové vzory je možné klasifikovat podle řady kritérií, například autoři publikace (Suarez-Figueroa, 2007) rozlišují logické, obsahové a architekturní vzory – první dva z těchto typů byly už v předchozích kapitolách vyloženy.

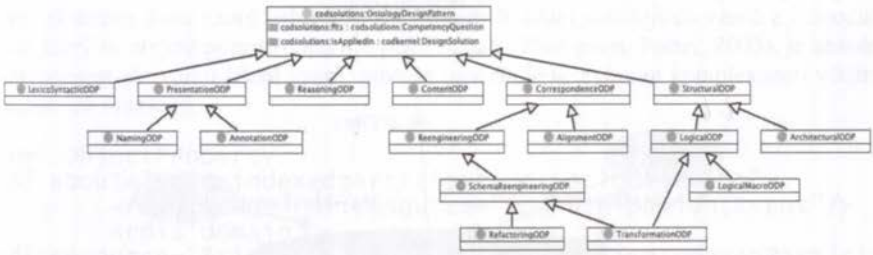
Existuje však celá řada dalších typů vzorů v rámci různých třídění. Samotné *návrhové* vzory jsou jen jednou ze skupin ontologických vzorů, přičemž Blomqvistová (Blomqvist, 2005) rozlišuje tyto typy ontologických vzorů:

- Aplikační vzory (Application patterns) – týkající se účelu, rozsahu, použití a kontextu implementovaných ontologií, včetně rozhraní a vztahů k jiným systémům.
- Vzory architektur (Architecture Patterns) – jež popisují, jak kombinovat či aranžovat implementované návrhové vzory tak, aby byl splněn účel celé ontologie.
- Návrhové vzory (Design patterns) – seskupení sémantických vzorů, které dohromady tvoří běžné a obecné součásti pro vývoj ontologií.
- Sémantické vzory (Semantic patterns) – na jazyku nezávislé popisy určitého pojmu, relace nebo axiomu.
- Syntaktické vzory (Syntactic patterns) – na jazyku závislé způsoby sestavování reprezentačních symbolů, aby byl zachycen určitý koncept, relace nebo axiom.

Portál *OntologyDesignPatterns.org* rozlišuje kategorie vzorů podle obr. 4.3. Vidíme zde, směrem odleva:

- *Lexikálně syntaktické vzory* zachycující vazbu mezi strukturami ontologie (včetně různých typů logicko-strukturních a obsahových vzorů) a jejich ekvivalenty v přirozeném jazyce. Tento typ vzorů lze použít pro účely generování textových souhrnů ontologií nebo naopak pro extrakci ontologických znalostí z volného textu. Příkladem je jeden ze vzorů pro vymezení vztahu podtřídy: *[A/any] NP<subclass> be [a/any] NP<superclass> PREP [(NP<class>,)* and/or NP<class*, kterému vyhovuje například věta: „A vegetarian pizza is a pizza without fish or meat." Zde nejde o ontologický vzor v pravém smyslu, ale o propojení ontologických vzorů na přirozený jazyk.
- *Prezentační vzory*, týkající se struktury pojmenování entit a hodnot jejich anotačních vlastností. Se jmennými vzory jsme se setkali v odst. 4.2.2.

- *Odvozovací vzory*, které se opět netýkají struktury ontologie jako takové, ale odvozovacích kroků nad ní.
- *Obsahové vzory*, se kterými jsme se setkali v odst. 4.3.2.
- Rozsáhlý konglomerát *korespondenčních a strukturních vzorů* zahrnující kromě logických vzorů (viz odst. 4.2.4) i vzory *mapovací* (AlignmentODP), *transformační*, *architekturní* atd.



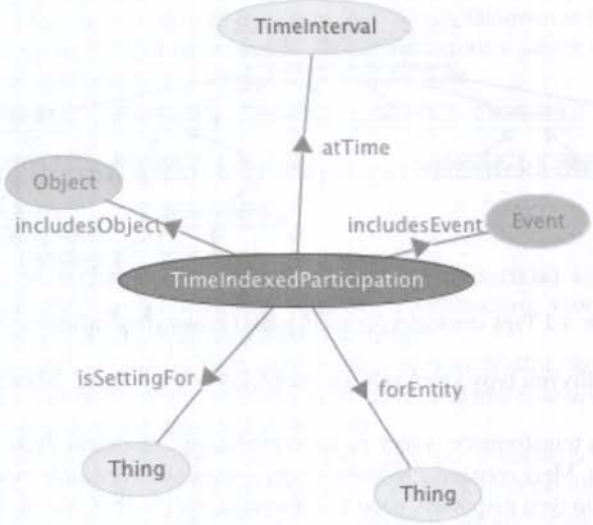
Obr. 4.3 Typy ontologických vzorů podle OntologyDesignPatterns.org.

Mezi jednotlivými typy vzorů jsou zjevné překryvy a interakce. Některé z nich si zde naznačíme:

- Mapovací a transformační vzory de facto představují propojení dvou vzorů logicko-strukturních. Mezi entitami z jednoho vzoru a druhého vzoru bude existovat propojení, a to buďto typu mapování, nebo transformace, viz (Šváb-Zamazal, Svátek, 2010).
- Vzory architektur představují vlastně logicko-strukturní vzory „ve velkém".
- Logicko-strukturní vzory nemusí být omezeny jen na abstraktní proměnné, ale mohou zahrnovat i jmenný aspekt. Zdá se, že v případě ontologií technických oborů, jejichž pojmy jsou hojně popsány víceslovnými termíny, lze pomocí takto „obohacených" vzorů do jisté míry vysledovat i problémy výchozí ontologické konceptualizace. Jde zejména o záměnu relace „část-celek" nebo jiné specifičtější relace za relaci generalizace (Šváb-Zamazal, Svátek, 2008), která se projeví neshodou mezi řídicími členy jmenných frází označujícími třídu a její podtřídu. Příkladem z reálné ontologie vojenského letectví je neshoda mezi třídou „AirspaceControlMeasure" (zhruba „opatření pro řízení letového prostoru") a její podtřídou „AirCorridor" („letový koridor") – speciálním typem opatření je zřejmě *zřízení* koridoru spíše než koridor samotný jako objekt vymezený prostorově. Lze očekávat, že mezi pojmy „measure" a „corridor" nebude v tezaurovém zdroji typu WordNet nalezena taxonomická souvislost.
- Komplexní příklad interakce mezi různými typy vzorů je ukázán v práci (Svátek, Šváb-Zamazal, Vacura, 2010). Jde o řešení problému importu obsahového vzoru do rozpracované ontologie, která proto musí být částečně transformována s využitím transformačních vzorů, přičemž jednotlivé varianty transformace vedou na cílové struktury odpovídající standardizovaným logicko-strukturním vzorům.

4.4.2 Příklad implementovaného vzoru

Portál *OntologyDesignPatterns.org* obsahuje u většiny obsahových vzorů nejen reprezentaci ve formě diagramu a slovního popisu, ale i formální definici v jazyce OWL. Příkladem může být vzor *časově indexované participace* (v grafickém schématu na obr. 4.4) související se vzorem, který byl už představen v odst. 4.3.2.



Obr. 4.4 Vzor časově indexované participace z portálu *OntologyDesignPatterns.org*.

Popis vzoru je vytvořen v šabloně sémantického CMS, na kterém je portál postaven – jde o šablonu specificky navrženou pro obsahové vzory, každý typ vzoru má totiž šablonu vlastní. Všechny informace jsou však následně transparentně uloženy do jediného souboru obsahujícího vzory v OWL/RDF (serializace) a umístěného na adrese http://ontologydesignpatterns.org/cp/owl/timeindexedparticipation.owl. Anotační vlastnosti v hlavičce souboru pak obsahují (kromě standardních vlastností rdfs:label, rdfs:comment, owl:versionInfo a rdfs:isDefinedBy) všechny další prvky šablony definované pomocí anotační metaontologie.[10] Jde o lokálně definované vlastnosti *hasAuthor*, *scenarios*, *hasConsequences*, *hasUnitTest*, *hasComponent*, *coversRequirements*, *hasIntent*, *isSpecializationOf* a *relatedCPs*. Význam většiny vlastností je asi zřejmý. Poznamenejme, že požadavky na obsahové vzory můžeme chápat jako kompetenční otázky definované pro takovou mikroontologii – zde jimi mohou být například: „Kdy se cosi účastnilo určité události? V jakém časovém období se události účastnil nějaký subjekt?". Jednotkový test (hodnota *hasUnitTest*) má v případě obsahových vzorů charakter SPARQLovských dotazů (jako operacionální verze kompetenčních otázek), zde se jedná o dotaz

[10] Umístěné na adrese http://www.ontologydesignpatterns.org/schemas/cpannotationschema.owl.

SELECT ?e ?o ?t WHERE {?x :includesEvent ?e . ?x :includesObject ?o . ?x :atTime ?t }.

Hlavní část souboru pak obsahuje logickou definici vzoru v jazyce OWL. Uveďme si některé jeho hlavní fragmenty. Dále uvedený kód definuje vlastnost *includesEvent*; kromě axiomů pro definiční obor a obor hodnot (viditelných v grafu) obsahuje i vztah k inverzní vlastnosti a k nadřazené vlastnosti, která je převzata z obecnějšího vzoru pro *časově indexovanou situaci*. Z jiného obecnějšího vzoru, definujícího *participaci*, je převzata třída *Event*. Vidíme tedy, že vzor pro časově indexovanou participaci je de facto složením dvou vzorů jednodušších. Princip skládání složitějších vzorů z jednodušších, který se zřejmě poprvé objevil v práci (Clark, Thompson, Porter, 2003), je zásadní pro možnost aktivního šíření vzorů, protože umožňuje se vyhnout komplexnosti většiny základních ontologií

```
<owl:ObjectProperty
rdf:about="&timeindexedparticipation;includesEvent">
        <rdfs:range rdf:resource="&participation;Event"/>
        <rdfs:domain
rdf:resource="&timeindexedparticipation;TimeIndexedParticip
ation"/>
        <owl:inverseOf
rdf:resource="&timeindexedparticipation;isEventIncludedIn"/
>
        <rdfs:subPropertyOf
rdf:resource="&timeindexedsituation;forEntity"/>
    </owl:ObjectProperty>
```

Analogicky je definována vlastnost *includesObject* i obě vlastnosti ke zmíněným dvěma inverzní (*isObjectIncludedIn*, *isEventIncludedIn*).

Ústřední třída celého vzoru, tj. „reifikovaná participační relace" *TimeIndexedParticipation*, je pak definována jako třída objektů zapojených do alespoň jednoho vztahu typů *includesObject*, *includesEvent*, *hasTime* a do alespoň tří vztahů typu *forEntity*.

```
    <owl:Class
rdf:about="&timeindexedparticipation;TimeIndexedParticipati
on">
        <rdfs:subClassOf
rdf:resource="&timeindexedsituation;TimeIndexedSituation"/>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty
rdf:resource="&timeindexedparticipation;includesObject"/>
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinalit
y>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty
rdf:resource="&timeindexedsituation;atTime"/>
```

```
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinalit
y>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty
rdf:resource="&timeindexedsituation;forEntity"/>
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">3</owl:minCardinalit
y>
            </owl:Restriction>
        </rdfs:subClassOf>
        <rdfs:subClassOf>
            <owl:Restriction>
                <owl:onProperty
rdf:resource="&timeindexedparticipation;includesEvent"/>
                <owl:minCardinality
rdf:datatype="&xsd;nonNegativeInteger">1</owl:minCardinalit
y>
            </owl:Restriction>
        </rdfs:subClassOf>
    </owl:Class>
```

Ostatní vlastnosti a třídy v rámci vzoru jsou nově deklarovány nebo přepoužity bez dodatečných logických podmínek, takže je uvádět nebudeme. V kódu byly dále vynechány anotační vlastnosti, které slouží pouze k doplnění lexikální informace určené pro člověka a technologických odkazů z entity na ontologii, ve které je definována.

4.4.3 Metodika eXtreme Design

Vzhledem k rostoucímu počtu návrhových vzorů je pro tvůrce ontologie často nesnadné vybrat z dostupných ty nejvhodnější. Proto vznikla metodika efektivního návrhu ontologií pomocí vzorů, nazvaná eXtreme Design (XD) (Blomqvist et al., 2010), na základě analogie s tzv. extrémním programováním.

Metodika předpokládá, že cílem je navrhnout relativně malou a kompaktní ontologii řešící spíše zadanou úlohu než vyčerpávajícím způsobem pokrýt určitou doménu. Někdy se pro takový typ ontologie používá přímo termín „ontologie zaměřená na úlohu" v kontrastu s tradičnějšími doménovými ontologiemi zaměřenými „na pokrytí", pro jejichž vývoj existuje řada metodik zmíněných v kapitole 1. Na obecné rovině metodika XD přejímá řadu aspektů extrémního programování, jako je práce ve dvojicích, časté testování, průběžná integrace apod. V souladu s tradičními postupy ontologického inženýrství se ovšem testování opírá o kompetenční otázky, na začátku formulované slovně, ale následně implementované formou dotazů v jazyce SPARQL.

Postup práce podle metodiky XD se rozpadá na několik kroků s řadou iterací:

1. Návrhář se seznámí se širším kontextem projektu a s věcnými experty.
2. Požadavky na ontologii jsou od expertů získány formou vzorových „příběhů" vyjadřujících fakta, která by cílová aplikace měla být s pomocí navržené ontologie zís-

kat. Každý „příběh" obsahuje jednu nebo několik málo vět textu, prioritu vyřešení a vazbu na jiné „příběhy".

3. Dvojice návrhářů dostane přidělený „příběh" k řešení a vytvoří si pro něj stránku wiki. Příběh je postupně transformován do textových *kompetenčních otázek*. Ty již nehovoří o konkrétních entitách z příkladu, ale zobecňují je na celé třídy. Vedle toho jsou shromážděna kontextová tvrzení, např. typu „každé X obsahuje alespoň jedno Z".

4. Návrháři se soustředí na určitou kompetenční otázku a snaží se pro ni nalézt odpovídající *obsahové vzory*, které by ji vyřešily. Například pro kompetenční otázku „Které objekty se účastní určité události?" mohou být kandidáty vzory týkající se participace, viz odstavce 4.3.2 a 4.4.2.

5. Pokud zvolené vzory věcně odpovídají, ale jsou příliš obecné nebo neúplné, aplikují se na ně operace *specializace* nebo *skládání*. V odstavci 4.4.2 jsme například viděli, že vzor časově indexované participace je složením dvou jednodušších vzorů, (obyčejné) participace a časově indexované situace. O specializaci hovoříme tehdy, když vznikne nový vzor, jehož entity jsou podřazeny (subClassOf, subPropertyOf) entitám vzoru původního. Z převzatých a upravených vzorů vznikne prvotní ontologický modul.

6. Provede se *jednotkové testování*: kompetenční otázka je převedena do formy dotazu SPARQL, znalostní báze je naplněna instancemi vytvořenými z obsahu původního příběhu, dotaz je následně položen a jeho výsledky zhodnoceny. Testování je pečlivě zdokumentováno na stránce wiki daného příběhu.

7. Když jsou všechny kompetenční otázky pro daný příběh vyřešeny, příslušný modul je publikován pod určitým URI, a tak zpřístupněn ostatním členům týmu, zejména pak dvojici zodpovědné za integraci celé ontologie.

8. Proběhne integrace spojená s dalšími testy. Při integraci jsou do ontologie začleněny i formalizované verze znalostí odpovídajících kontextovým tvrzením shromážděným ve 3. kroku. Otestovaná ontologie je nakonec zveřejněna jako celek.

Metodika XD je v současnosti podpořena i specializovaným nástrojem (nazvaným XDTools), implementovaným jako rozšíření obecného ontologického editoru NeOn Toolkit.[11] Hlavním výzkumným problémem podmiňujícím širší využívání metodiky je automatická podpora vyhledávání relevantních vzorů ve 4. kroku. Vedle plnotextového vyhledávání se předpokládá sémantické vyhledávání za pomoci metaontologie pro oblast ontologických návrhových vzorů jako takovou.

4.5 Diskuse a shrnutí

Je zřejmé, že pojem „lepšího" či „horšího" obsahu ontologií je – ve srovnání s pojmy, jako je formální konzistence ontologií nebo výpočtová složitost odvozovacích algoritmů – daleko subtilnější a subjektivnější.

Častým omylem začínajících vývojářů aplikací sémantického webu bývá, že pečlivé zpracování specifikací ontologického jazyka a jeho podporu odvozovacími nástroji považují za postačující podmínku pro vznik v praxi využitelných ontologií. Jak jsme viděli v této kapitole, ve skutečnosti tomu tak není. Ontologický artefakt, který je z formálního hlediska zcela v pořádku, může být zcela neadekvátní z hlediska obsahu. Problém ade-

[11] To lze částečně řešit pomocí rozšíření jazyků, které podporují neurčitost, viz http://www.neon-toolkit.org

kvátnosti ontologického obsahu na sémantickém webu bychom mohli, bez nároku na úplnost, zobecnit na tři klíčová hlediska (Svátek, 2004):

- *Přesnost*: Obsah ontologie by měl co nejvěrohodněji odpovídat situacím v reálném světě. Mnoho ontologických modelů je oproti realitě natolik zkreslených či zjednodušených (ať už vinou nedostatečně pečlivé analýzy reality, nebo kvůli omezením reprezentačního jazyka), že formálně správná odvození nad nimi vedou k nesmyslným závěrům.
- *Srozumitelnost*: Ontologie by měla být co nejlépe srozumitelná lidem, kteří se na jejím vývoji nepodíleli, což je zvláště významné v otevřeném webovém prostředí. Zde má velký význam vhodné pojmenování prvků ontologie, ale také modularizace na části, které je člověk schopen rozumově obsáhnout.
- *Způsobilost k odvozování*: Nad ontologií (pokud má mít vyšší ambice než tradiční tezaury a klasifikační schémata, což v prostředí sémantického webu lze předpokládat) by mělo být možné netriviálně strojově odvozovat. Pro toto je třeba využít možností formálního aparátu jazyka.

Mezi těmito hledisky lze vypozorovat různé pozitivní i negativní vazby. Snaha o bohaté odvozovací možnosti nad ontologií někdy vede k přidávání vztahů, které platí pouze „modelově" a nikoliv paušálně[12], čímž se snižuje přesnost. Podobně i snaha o maximální srozumitelnost pro nezasvěcené uživatele může vést ke snižování přesnosti (např. pokud se přesné, avšak málo známé pojmy nahradí běžnými ale nepřesnými). Na druhé straně vysoká míra formalizace (pro potřebu odvozování) může napomoci odhalit již ve fázi úvodního návrhu některé odchylky, např. taxonomické kostry, oproti realitě.

Ontologické inženýrství, původně vzniklé jako pomocná disciplína v kontextu umělé inteligence a výpočtové logiky, se zejména po r. 2000 etablovalo jako samostatný obor zahrnující velký počet principů, metod a technologických nástrojů. Ústředním problémem je hledání kompromisu mezi potřebou rozšířit množinu tvůrců ontologií mimo nepočetný okruh odborníků na logiku a filosofii (tj. zejména mezi běžné softwarové inženýry a doménové experty) a požadavky na vysokou kvalitu ontologií kladených zvláště aplikacemi vyžadujícími strojové odvozování. Skutečností je, že ontologií, které by byly věcně přesné, lidsky srozumitelné a zároveň způsobilé pro netriviální strojové odvozování, je prozatím velmi málo. K tomu, aby sémantický web, ale i znalostní technologie obecně, mohly dostát svým deklarovaným možnostem, je široké povědomí o metodách a nástrojích ontologického inženýrství nezbytností.

Největší pozornost se v současnosti zřejmě soustřeďuje na metody založené na návrhových vzorech, které jsou i těžištěm celé kapitoly. Mezinárodní iniciativa spojená s portálem *OntologyDesignPatterns.org* usiluje o shromáždění rozsáhlé kolekce zahrnující jak vzory obsahové, tak i logické, ale vedle toho i další, specifičtější typy vzorů, jako jsou vzory architektur (pro návrh páteřní struktury ontologie jako celku) či transformační (pro převod mezi různými ontologickými jazyky či modelovacími styly). Významné je, že pro ontologické inženýrství v kontextu rozsáhlého a heterogenního prostředí sémantického webu již vznikají specializované metodiky včetně softwarové podpory. Nejnáročnějším úkolem je zde zajistit mechanismus, který vývojáři ontologie na základě vágního zadání uživatelské potřeby vyhledá adekvátní vzory. Aktuální výzkumné trendy proto směřují k provázanosti ruční tvorby ontologií s technikami vyhledávání ontologií a vzorů, mapování a integrace ontologií (o obojím byla řeč v kapitole 2) a v neposlední

[12] To lze částečně řešit pomocí rozšíření jazyků, které podporují neurčitost, viz http://www.neon-toolkit.org.

řadě také k získávání fragmentů ontologií dolováním z textů (Maedche, 2002), (Buitelaar, Cimiano, Magnini, 2005), (Cimiano, 2006).

Vzhledem k intenzivnímu rozšiřování okruhu propojených datových zdrojů (Linked Data) se jako aktuální jeví také „reaktivní" model ontologického inženýrství pro tento typ dat. Linked Data jsou zpravidla založena na proprietárních schématech specifických pro každý datový zdroj, s výjimkou sdílení několika málo populárních slovníků (vCard, FOAF, Dublin Core apod.). Nabízí se možnost mapovat tato obvykle strukturně chudá a z ontologického hlediska ne vždy ideálně navržená schémata na propracovanější ontologické modely, např. typu obsahových návrhových vzorů. Vedle toho se uplatňují metody „dolování z Linked Data" (Lehmann, 2011), (Völker, 2011), jejichž cílem je naučit se induktivně axiomy schématu z existujících dat.

Literatura

Allemang D., Hendler J.: *Semantic Web for the Working Ontologist.* Morgan Kaufmann, 2008.

Blomqvist E., Sandkuhl K.: Patterns in ontology engineering: Classification of ontology patterns. In: *Proceedings of the 7th International Conference on Enterprise Information Systems*. Miami, May 2005.

Blomqvist E., Presutti V., Daga E., Gangemi A.: Experimenting with eXtreme Design. In: *EKAW 2010*, s. 120–134.

Buitelaar P., Cimiano P., Magnini B. (eds): *Ontology Learning and Population.* IOS Press, 2005.

Buitelaar P., Cimiano P., Haase P., Sintek M.: Towards Linguistically Grounded Ontologies. In: *Proceedings ESWC*, 2009, s. 111–125.

Cimiano P.: *Ontology Learning and Population from Text.* Springer, 2006.

Clark P., Thompson J., Porter B.: Knowledge Patterns. In: S. Staab, R. Studer (eds): *Handbook of Ontologies, pages.* Springer, Berlín 2003, s. 191–207.

Espinoza M., Gómez-Pérez A., Montiel-Ponsoda E.: Multilingual and Localization Support for Ontologies. In: *Proceedings ESWC*, 2009, s. 821–825.

Gangemi A.: Ontology Design Patterns for Semantic Web Content. In: *Proc. of International Semantic Web Conference 2005*, s. 262–276.

Horridge M.: *Protégé OWL Tutorial. Verze 1.3.* University of Manchester, 2011. http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/

Lehmann J., Auer S., Bühmann L., Tramp S.: Class expression learning for ontology engineering, *J. Web Semantics*, 9(1), 2011, s. 71–81.

Maedche A.: *Ontology Learning for the Semantic Web*. Kluwer Academic Publishers, 2002.

Masolo C., Borgo S., Gangemi A., Guarino N., Oltramari A.: *Ontology library*. WonderWeb Deliverable no. 18, 2002. http://wonderweb.semanticweb.org/deliverables/D18.shtml

Motik B., Patel-Schneider P. F., Parsia B.: *OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax.* W3C Recommendation 27, October 2009. http://www.w3.org/TR/2009/REC-owl2-syntax-20091027/

Noy N., McGuinness D.: *Ontology Development 101: A Guide to Creating Your First Ontology.* Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

Noy N., Rector A.: *Defining N-ary Relations on the Semantic Web: Use with Individuals*. W3C Working Draft 21, July 2004. http://www.w3.org/TR/swbp-n-aryRelations

Noy N., Uschold M., Welty C.: *Representing Classes As Property Values on the Semantic Web*. W3C Working Group Note 5 April 2005. http://www.w3.org/TR/swbp-classes-as-values/

Rector A. L. et al.: OWL Pizzas: Practical Experience of Teaching OWL-DL: Common Errors & Common Patterns. *Proc. EKAW 2004*, s. 63–81.

Schober D. et al.: Survey-based naming conventions for use in OBO Foundry ontology development. *BMC Bioinformatics*, Vol.10, Issue 1, 2009.

Suarez-Figueroa M. C., Brockmans S., Gangemi A., Gomez-Perez A., Lehmann J., Lewen H., Presutti V., Sabou M.: *NeOn modelling components. NeOn deliverable D5.1.1*. Universidad Politecnica de Madrid, 2007.

Sunagawa E., Kozaki K., Kitamura Y., Mizoguchi R.: A Model of Roles in Ontology Development Tool: Hozo. In: *Proc. EKAW 2006*, LNCS, Springer-Verlag, 2006.

Svátek V.: Design Patterns for Semantic Web Ontologies: Motivation and Discussion. In: *7th Conference on Business Information Systems*, Poznaň 2004.

Svátek V., Šváb-Zamazal O., Presutti V.: Ontology Naming Pattern Sauce for (Human and Computer) Gourmets. In: *Workshop on Ontology Patterns at ISWC'09*. Washington DC, 2009.

Svátek V., Šváb-Zamazal O., Vacura M.: Adapting Ontologies to Content Patterns using Transformation Patterns. In: *WOP 2010 – Workshop on Ontology Patterns, at ISWC 2010*.

Šváb-Zamazal O., Svátek V.: Analysing Ontological Structures through Name Pattern Tracking. In: *EKAW 2008*, Acitrezza, Itálie, 2008.

Šváb-Zamazal O., Svátek V., Iannone L.: Pattern-Based Ontology Transformation Service Exploiting OPPL and OWL-API. In: *EKAW 2010*, Lisabon 2010.

Völker J., Niepert M.: Statistical Schema Induction. In: *ESWC* (1), 2011, s. 124–138.

5 Aplikace ontologií v průmyslové výrobě

5.1 Úvod

Průmyslová výroba a její řízení se stávají čím dál tím více závislé na distribuovaných softwarových řešeních založených na filosofii agentů či holonů. Takovéto systémy se opírají o množiny relativně autonomních subsystémů, jež umožňují optimalizovat výkon řízeného celku a současně v případě potřeby adekvátně rychle reagovat na nepředvídané nebo jen řídce se vyskytující situace. Typické (a nezbytné) pro takovéto systémy je, že využívají sémantickou informaci či ontologickou znalost, která je udržována v dobách klidného, na řídicí funkcionality časově nenáročného provozu tak, aby mohla být její aktuálně platná podoba v plné síle využita lokálně k řešení problému, jenž vyžaduje mimořádně rychlou řídicí odezvu (Mařík, McFarlane, 2005). Tedy sémantika situace a ontologické znalosti patří do výbavy elementárních řídicích jednotek, které jsou často typu *holon* či *agent*.

Velmi náročným (časově i finančně) bývá návrh a implementace agentových či holonických řídicích systémů. Nicméně dle povahy úlohy je u jednotlivých agentů či holonů v té či oné míře možné opakovat, tedy využít znovu, kód, který již byl u podobných jednotek v témže nebo svým zaměřením podobném agentním systému použit. Je však třeba potřebný kód efektivně vyhledat a ev. po modifikaci znovu reimplementovat u dalších jednotek. I k vyhledávání kódů nebo jejich částí lze účinně využít ontologickou či sémantickou znalost.

Využívání sémantických či ontologických znalostí za podpory technologie sémantického webu – společně s viditelnou tendencí k využívání technologií SOA (Service Oriented Architecture) – se v současné době stává přirozeným trendem při budování rozsáhlejších systémů k řešení průmyslových úloh. Technologie sémantického webu se opírá o vizi společného informačního rámce či prostředí, jež umožňuje strojům vyhledávat, interpretovat a sdílet data na webu. K dosažení této vize je však nezbytné, aby informace byla na webu sémanticky strukturována. Používaná terminologie není zatím jednotně vžitá, často je míněno pojmy sémantika, metadata či ontologie jedno a totéž. Obecně lze konstatovat, že sémantika je vyjadřována *ontologií*, jež poskytuje slovník pro popis konceptů, jejich vzájemných relací a případných omezení v dané oblasti zájmu.

Ontologie se čím dál tím více využívají pro sémantický popis znalostí vztahujících se k výrobním systémům (Runde, Fay, 2011). Obecně řečeno, ontologie popisuje určitou konkrétní oblast formou tříd objektů (těmito objekty mohou být stroje, nástroje, operace, zakázky, úlohy, části výrobků, dělníci atd.) a jejich vzájemných relací. Zatímco dříve byla interpretace sémantické informace napevno zaprogramována do kódu jednotlivých agentů, dnes se agenti spíše opírají o ontologické znalosti vyjádřené v explicitní podobě a ty pak příslušným způsobem dle své potřeby interpretují. To je významný pokrok. Neboť agenti tak získávají významnou možnost sémantickou znalost využívat efektivně-

ji, přemýšlet o ní, vyvozovat nad ní závěry v podobě nových faktů či hypotéz. Nejnovější průmyslové agentní systémy jsou navrhovány tak, že mohou pracovat s ontologiemi měnícími se v reálném čase bez jakékoliv potřeby měnit program agenta. Například ontologie může být rozšířena o nový typ výrobku a plán jeho výroby či o informaci o novém výrobním stroji v lince. Následně jsou agenti schopni automaticky vstřebat nové znalosti v ontologii a zabezpečit výrobu nového produktu či v druhém případě začít využívat možností a schopností nového stroje pro efektivnější výrobu stávajících výrobků, a to bez jakékoliv potřeby cokoliv v řídicím systému přeprogramovávat. Sémantická informace je velmi vhodná i pro potřeby extrémně rychlého řešení rekonfiguračních úloh (Vrba, Mařík, 2010).

Aplikace ontologií a relevantních technologií, zejména technologií sémantického webu, byly již alespoň ilustrativně uvedeny v předchozích kapitolách. Cílem této kapitoly je představit vybrané aplikace v konkrétní doméně – v oblasti průmyslové výroby. Tato doména je specifická tím, že je těsně spojena s fyzickým světem. To vyvolává řadu specifických požadavků. Aplikace musí být schopna reagovat na události ve fyzickém světě, a to často v reálném čase. Při hledání informací jakožto podkladů pro rozhodování není v některých doménách problémem částečná nepřesnost nebo dlouhý čas potřebný pro hledání. Selhání řídicí aplikace v těchto ohledech ovšem může vést k fyzickému poškození nákladného průmyslového vybavení. Doména průmyslové výroby ale samozřejmě není v těchto omezeních jediná – například pro elektronický obchod také obyčejně nechceme nepřesné informace o ceně.

Kapitola je organizována následujícím způsobem: V oddílu 5.2 je popsána reálná úloha vyhledávání relevantních částí kódu ve strukturovaných datech. Zde přiblížíme problematiku hledání za použití technologií sémantického webu na konkrétních datech z průmyslové výroby. Oddíl 5.3 je věnován problematice využívání sémantiky v holonických a multiagentních průmyslových řídicích systémech na obecné přehledové úrovni, a to včetně představení nejběžnějších přístupů. Podrobnější popis návrhu obecné ontologie pro potřeby multiagentních systémů v automatizaci je uveden v oddílu 5.4. Představen je mimo jiné i způsob integrace ontologie do průmyslového řídicího systému, který ve svém důsledku částečně ovlivňuje návrh softwarové architektury agentů. Dvě rozsáhlejší konkrétní případové studie, opírající se použití sémantické informace, na jejichž řešení se autoři kapitoly významně podíleli, jsou prezentovány v oddílu 5.5. V těchto úlohách se využívá sémantického rozšíření simulačního nástroje MAST (Vrba, 2006) a je zde ukázáno, že použití ontologií v holonických a multiagentních průmyslových řídicích systémech umožňuje, aby systém byl jednoduše rozšiřitelný i za běhu a aby byl schopen se adaptovat i na změny, které by bez použití ontologií vyžadovaly přeprogramování nezanedbatelné části systému. Závěrečné shrnutí poskytuje oddíl 5.6.

5.2 Hledání ve strukturovaných datech

Jak již bylo uvedeno v předchozích kapitolách, jednou z důležitých aplikací ontologií je hledání v datech. Motivací pro realizaci sémantického webu je zpřístupnit data takovým způsobem, aby i počítače byly schopny data vyhledávat, integrovat a usuzovat z nich. Jak jsme již uvedli, původní důraz sémantického webu byl na zpracování informací v přirozeném jazyce a na jejich anotování, nicméně větší praktický ohlas vzbudilo gene-

rování a sdílení strukturovaných dat z databází. V podobném duchu se nese i aplikace RoSS (Rockwell Structural Search), popsaná v této kapitole, kde rozvádíme strukturální hledání v datech z návrhu, provozu a údržby automatického řídicího systému. Aplikace RoSS byla vyvinuta ve výzkumném středisku firmy Rockwell Automation v Praze a ukazuje, jak s použitím technologií sémantického webu klást a zodpovídat i netriviální dotazy, které bychom s prostým fulltextovým hledáním nebyli schopni ani formulovat.

5.2.1 Popis domény

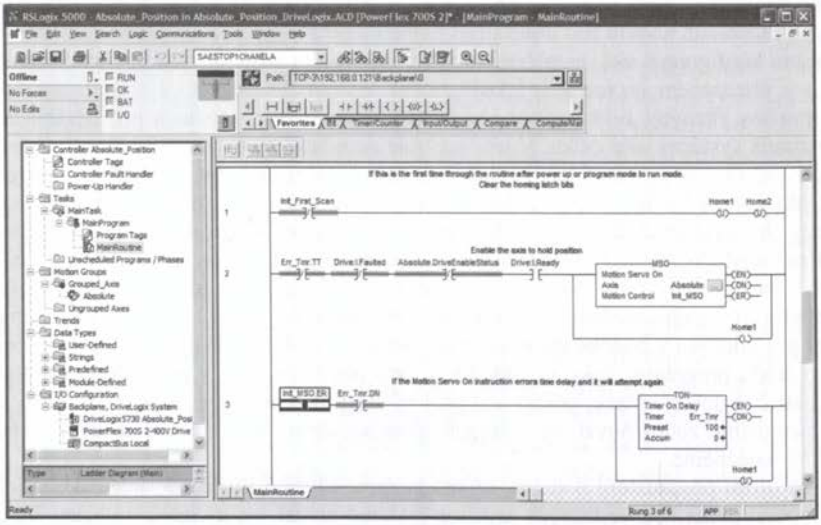
Návrh, provoz a údržba automatického řídicího systému, například výrobní linky, je určitě znalostně velmi náročný úkol. Celý systém je popsán daty různého charakteru – typickým řídicím kódem pro programovatelné automaty, popisem rozhraní člověk–stroj, popisem konfigurace sítě, naměřenými daty, záznamem událostí, dokumentací v podobě textu v přirozeném jazyce atd. I když různé součásti takovéhoto systému mohou být navrhovány různými lidmi, přece jen je pro celkové porozumění a pro údržbu nutné porozumět systému jako celku. V této kapitole detailněji rozebereme jen zdánlivě jednoduchý požadavek pro usnadnění celého procesu návrhu, provozu a údržby – schopnost v těchto datech hledat jako v celku, schopnost formulovat složitější dotazy a na tyto dotazy dostávat kvalifikovanou odpověď. I když máme k navrženému a pracujícímu systému dostupný veškerý kód a veškerou dokumentaci, je pro efektivní práci s kódem, např. i pro potřeby jeho eventuálního znovupoužití, žádoucí, aby existovala možnost účinného a rychlého hledání v tomto kódu. Tento problém určitě zná každý programátor, který pro údržbu i v podobě menších úprav nebo v podobě hledání chyb potřebuje orientaci v kódu programu. Také ví, nakolik je tato orientace těžší v rozsáhlejším systému, bez patřičné dokumentace, pokud kód psal někdo jiný a je-li kód psán ve více jazycích. Asi nemusíme zdůrazňovat, že v rozsáhlejším systému často alespoň některý z těchto faktorů nalezneme.

Pro bližší porozumění doméně, na níž budeme hledání ilustrovat, uvedeme nejdříve podrobněji popis dat, ve kterých zejména budeme hledat – konkrétně v oblasti řídicího kódu a oblasti rozhraní člověk–stroj. Aplikace pro hledání je stavěna pro potřeby produktů společnosti Rockwell Automation, na nichž bude též ilustrována, nicméně principy jsou samozřejmě platné obecně – podobným způsobem lze hledat v jakýchkoliv datech, kde je jasně daná jejich struktura.

5.2.1.1 Řídicí kód pro programovatelné automaty

Pro programování klasických programovatelných automatů (PLC) se v současnosti používají jazyky standardu IEC 61131-3, např. žebříčkový kód či správněji liniové schéma (relay ladder code), funkční bloky, sekvenční funkční grafy nebo strukturovaný text. Žebříčkový kód, který patří k nejoblíbenějším ve Spojených státech amerických, byl původně zaveden pro popis reléové logiky a je stále používán pro zakreslování elektrických či logických schémat. Název je odvozen od toho, že vizuálně připomíná žebřík – na okrajích jsou dvě stojny, mezi nimi jsou jakoby příčky žebříku, přičemž na každé příčce (rung) může být zapojeno několik logických prvků. Popularita tohoto vizuálního zápisu kódu vyplývá i z toho, že porozumění jemu není pro technika obtížné, nicméně samozřejmě existuje například i textová podoba zápisu stejného kódu. Tento kód se používá pro programování řídicích systémů pro životně kritické aplikace, od relativně jednoduchých po velmi složité.

Kód (viz obr. 5.1) si lze představit spíše jako pravidlový než procedurální jazyk. Příčka reprezentuje jedno pravidlo – pokud jsou splněny předpoklady uvedené v levé části, provede se akce uvedená v části pravé. Pokud je toto implementováno pomocí relé nebo jiných elektromechanických zařízení, je provádění jednotlivých pravidel současné a okamžité. Při implementaci v programovatelném automatu dochází k cyklickému procházení pravidel, a protože je toto prováděno velmi rychle, je dosažen efekt okamžité a současné reakce na procesy v řízeném systému. Jednotlivé instrukce mají navíc vedlejší efekty v podobě čtení a zápisu z paměťových míst i do nich a pravidla v programu jsou tak vzájemně provázána.



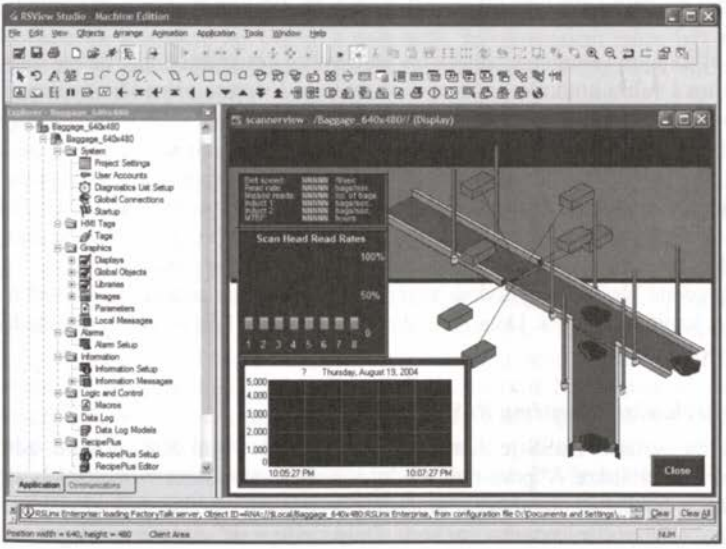
Obr. 5.1 Obrazovka programu RSLogix 5000 – editace žebříčkového kódu.

Různé podoby kódu jsou popsány ve standardu IEC 61131-3. Příkladem softwarového balíku pro programování v žebříčkovém kódu dle tohoto standardu je RSLogix 5000, který podporuje programovatelné automaty Allen-Bradley. Příklad editace kódu v tomto programu vidíme na obr. 5.1. Pro porozumění dalšímu textu shrňme, že projekt v balíčku RSLogix 5000 obsahuje zejména definici proměnných (tagů, pojmenovaných paměťových míst různého typu) a hierarchický souhrn úloh, programů a rutin, jež obsahují instrukce v podobě žebříčkového kódu.

5.2.1.2 Rozhraní člověk–stroj

Pro ovládání řízeného a řídicího procesu člověkem slouží takzvané rozhraní člověk–stroj (Human-Machine Interface – HMI). Tento podsystém zajišťuje vizualizaci průběhu procesu, možnost spouštění či zastavování procesu a je důležitý zejména pro diagnostiku chyb a pro uvedení systému do normálního běhu po poruše, například po poruše některého fyzického zařízení. Rozhraní je obvykle propojeno s jedním nebo i více programovatelnými automaty a fyzicky je tvořeno panelem s obrazovkou a klávesnicí nebo s dotykovou obrazovkou tak, aby operátor byl schopen proces sledovat a v případě potřeby ovládat.

Příkladem softwarového balíku pro návrh a realizaci takovéhoto systému je RSView Enterprise. Ilustraci návrhu obrazovky vidíme na obr. 5.2. Obrazovka se skládá z různých grafických a ovládacích prvků, jež jsou úzce provázány řídicí aplikací běžící na PLC. Těmito prvky jsou například text, zobrazení hodnot (tagů) přečtených z PLC, různé typy tlačítek, jejichž stisk generuje hodnoty, jenž jsou zapisovány do PLC, zobrazení průběhu sledované veličiny apod. Pro složitější aplikace lze doplnit program v podobě maker nebo kódu v programovacím jazyce. Obrazovek je většinou více a lze mezi nimi přecházet.



Obr. 5.2 Vzhled programu RSView Enterprise – editace obrazovky HMI.

5.2.2 Vyhledávání

Návrh řídicího systému výrobní linky sestává z více projektů vytvořených v uvedených, popř. i dalších, softwarových nástrojích. Kód je poté nahrán na programovatelné automaty a na ovládací panely. Všechny projekty jsou typicky vzájemně propojeny, mohou být navrhovány více inženýry, navíc existují ve více verzích (při aktualizacích musíme mít možnost návratu k předchozí funkční verzi). Postup při hledání problémů a při aktualizacích pak může i přes existující dokumentaci vyžadovat postupné otevírání všech projektů a jejich analýzu, což je samozřejmě velmi časově náročné. Možnost komplexního vyhledávání v projektech, ať již na počítačích nebo přímo v řídicím systému, by tento čas zkrátila. Hledání může probíhat v rozsahu jednoho souboru, na souborovém systému počítače, v síti počítačů, popř. v celém řídicím systému.

Pro objasnění motivace a cílů si před dalším popisem uveďme příklady dotazů:

- Najdi, kde se v žebříčkovém kódu vyskytují v jedné příčce instrukce XIO pro proměnné Err_Tmr.TT a Drive:I.Faulted.
- Najdi instrukci SUB, kde je jako operand použita konstanta 0,5.
- Najdi obrazovky, kde se vyskytuje textový objekt s červeným pozadím a s řetězcem „plnění".

- Najdi všechny grafické objekty typu sloupcového grafu, které mají nastaven práh na 1000, a ukaž, které proměnné zobrazují.
- Najdi instrukce v žebříčkovém kódu, které nastavují proměnnou, jež rozsvěcuje nápis „varování" na libovolné obrazovce v daném projektu.
- Najdi rutiny, které nějak používají proměnnou, jíž zobrazuje libovolný objekt typu měřidlo.

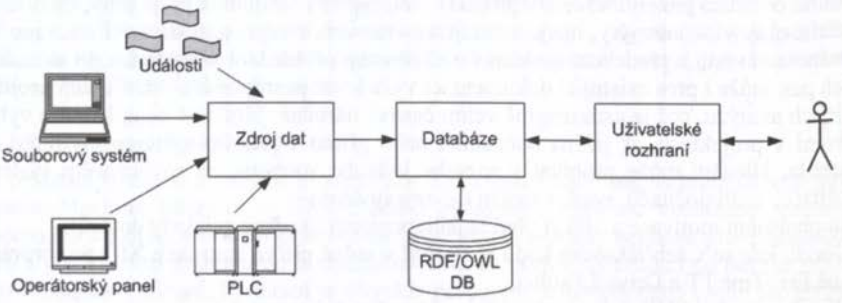
Všimněme si, že poslední dva dotazy pro své zodpovězení potřebují integraci dat ze dvou podoblastí, a to z žebříčkového kódu a z rozhraní člověk–stroj. Představme si, jakou práci by dalo najít tuto informaci postupem, při kterém bychom otevírali všechny projekty.

Dále si již nyní všimněme, že dotazy jsou velmi konkrétní a, jak už lze tušit, odpovídají i datům s velmi konkrétní a jasnou strukturou. V tomto případě tedy nemusíme provádět sémantické zjednoznačnění (disambiguaci) dotazu, tak jako je to potřeba v sémantickém hledání. Stejně tak anotování dat pro prohledávání může probíhat zcela automaticky, jejich struktura je pevně daná. Rozhodně alespoň v základní podobě anotování a hledání není třeba řešit problémy s přirozeným jazykem nebo s více ontologiemi pro popis téhož, tak jako je tomu na sémantickém webu (viz předchozí kapitoly). Proto také hledání v této podobě nazýváme *strukturální*, ne přímo *sémantické*.

Pro uvedené hledání s výhodou použijeme technologie sémantického webu, jak byly popsány v kapitolách 2 a 4. Data převedeme do formátu RDF/OWL a pro hledání použijeme jazyk SPARQL.

5.2.2.1 Architektura systému RoSS

Architektura systému RoSS je ilustrována na obr. 5.3. První část systému, zdroj dat, se stará o plnění databáze. V prvé řadě sleduje souborový systém v počítačích zapojených do hledání a informuje databázi o změně nebo novém obsahu, pokud dojde ke změně stávajícího nebo přidání nového souboru. Dále sleduje síť se zařízeními, jakou jsou programovatelné automaty, a opět při změně dat v zařízeních nebo při změně struktury sítě informuje databázi. V databázi jsou pak uložena sémanticky strukturovaná data spolu s ontologiemi, představujícími slovníky pro sémantickou anotaci. Tato data slouží nejen pro vyhodnocování dotazů, ale i pro jejich konstrukci. Uživatelské rozhraní umožňuje uživateli konstruovat dotazy a dostávat na ně odpovědi.



Obr. 5.3 Architektura systému RoSS.

5.2.2.2 *Převod dat do formátu RDF/OWL*

V aplikaci RoSS je použit takovýto postup konverze – z dat, tedy například z projektů popsaných v předchozím výkladu, získáme jejich popis v jazyce XML. Z dokumentů XML pak pomocí ontologie generujeme data ve formátu RDF a ukládáme je tak, aby bylo možné provádět hledání v integrovaných datech.

Jak již víme z kapitoly 2, jazyk RDF slouží pro popis (či vlastně přímo vyjádření) dat, zatímco jazyk OWL je určen pro popis ontologie. Ontologie zde popisuje obecnou možnou strukturu dat v projektu – vyjadřuje tedy, že například v doméně žebříčkové logiky existují rutiny, úlohy, příčky, instrukce, proměnné atd., dále že rutina může obsahovat instrukce, že instrukce má jméno a operandy, že operandem může být proměnná nebo konstanta, že jméno proměnné může být jen řetězcovým literálem, zatímco číslo příčky může být jen celočíselným literálem.

Protože máme pevně daný formát dat, můžeme ontologii též generovat automaticky – a to buď přímo z dokumentů ve formátu XML, nebo například z popisu XML Schematu. Protože ale generovaná ontologie obvykle nevyhovuje zamýšlenému formátu beze zbytku (je příliš svázaná s konkrétním formátem XML), po vygenerování ontologii před použitím ke generování dat a pro hledání manuálně upravujeme. Ontologie obsahuje například i informaci o tom, že relace „obsahuje" (např. pro vztah úloha obsahuje rutinu) je tranzitivní, a pak se tato vlastnost pro hledání též použije. Vlastní generování dat z dokumentů ve formátu XML je pak řízeno touto ontologií. Vstupní dokument je transformován do RDF, a pokud například pro nějakou značku nebo atribut v XML nenalezneme korespondenci v ontologii, přeskočíme ji. Podobně probíhá přejmenovávání a další transformace. Výsledkem jsou pak samozřejmě data, která dané ontologii odpovídají. Vše je pak společně s ontologiemi uloženo tak, aby bylo možné v integrovaných datech vyhledávat. Před uložením lze též dopočítat tranzitivní uzávěr všech logických důsledků vygenerovaného dokumentu v RDF a související ontologie jako například pro uvedenou tranzitivitu vztahu „obsahuje" – z důvodu praktických omezení při uvažování nad velkým objemem se nám tento postup osvědčil.

Mimo vlastní data projektu musíme též uchovávat informaci o umístění zdrojových dat – nejen pro vyhledávání, ale i pro pozdější aktualizace. Pro data ze souborového systému tak přidáváme informaci o jméně souboru a o jméně počítače a další informace, jako je velikost souboru, datum vytvoření a poslední modifikace, informaci o atributech a další metadata, která daný souborový systém umožňuje ukládat. Pro data z řídicích automatů nebo z operátorských panelů je nutné uložit adresu zařízení, například ve formě adresy IP.

Jen pro zajímavost uveďme, že souhrnná ontologie pro uvedené domény obsahuje 185 tříd a 770 datových vlastností. Třídy a vlastnosti jsou provázány přes definiční obory a obory hodnot. Mezi třídami je bohatá možnost provázání pomocí objektových vlastností, přes 2676 možných vazeb (tak vysoké číslo je dáno tranzitivitou tohoto provázání).

5.2.2.3 *Hledání pomocí jazyka SPARQL*

Nyní, když máme data uložená ve formátu RDF/OWL spolu se souhrnnou ontologií, můžeme klást strukturované dotazy pomocí jazyka SPARQL. S výhodou využijeme to, že máme v době dotazování dostupné i logické důsledky plynoucí z dat a z ontologií – dotazy pak mohou být jednodušší.

Uveďme si příklad dotazu „Najdi instrukci SUB, kde je jako operand použita konstanta 0,5". Pro dotazování v této podobě si musíme ujasnit, co se myslí příkazem „najdi instrukci" – řekněme, že nás zajímá jméno souboru, rutiny a číslo příčky. Dotaz pak vypadá takto:

```
PREFIX File: <http://rockwell.com/NS/File.owl#>
PREFIX RSLogix: <http://rockwell.com/NS/RSLogix.owl#>
PREFIX Common: <http://rockwell.com/NS/Generic.owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

SELECT * WHERE {
[ a File:File;                              # hledám soubor
  File:name ?fileName;                      # zajímá mě jeho jméno
  Common:contains                           # musí obsahovat
  [ a RSLogix:Routine;                      # rutinu
    RSLogix:name ?routineName;
    Common:contains
    [ a RSLogix:Rung;                       # ...příčku
      RSLogix:number ?rungNumber;
      Common:contains
      [ a RSLogix:Instruction;              # ... instrukci
        RSLogix:name "SUB"^^xsd:string;
        Common:contains
        [ a RSLogix:Operand;              # ... a konečně operand 0,5
          RSLogix:value "0.5"^^xsd:string
] ] ] ] ] }
```

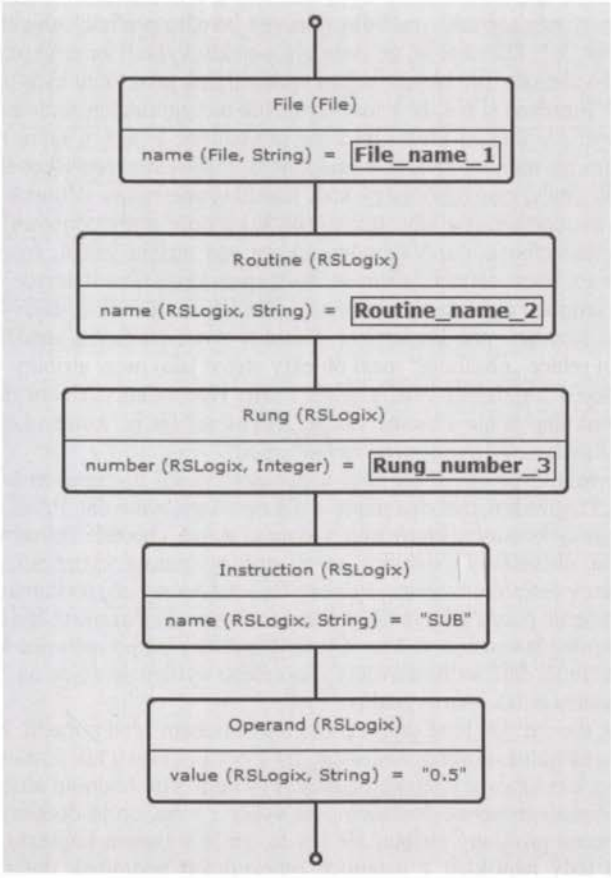
Jako výsledek dostaneme seznam odpovídajících jmen souborů a rutin a čísel příček. Jiným příkladem je dotaz kombinující dvě oblasti: „Najdi rutiny, které nějak používají proměnnou, kterou zobrazuje libovolný objekt typu měřidlo".

```
PREFIX File: <http://rockwell.com/NS/File.owl#>
PREFIX RSLogix: <http://rockwell.com/NS/RSLogix.owl#>
PREFIX RSView: <http://rockwell.com/NS/RSView.owl#>
PREFIX Common: <http://rockwell.com/NS/Generic.owl#>

SELECT * WHERE {
[ a RSView:Gauge;                       # měřidlo
  Common:contains
  [ a RSView:Tag;                       # zobrazuje proměnnou
    RSView:name ?tagName                # s tímto jménem
] ] .
[ a File:File;                          # hledám soubor
  File:name ?fileName;
  Common:contains
  [ a RSLogix:Routine;                  # ... s rutinou
    RSLogix:name ?routineName;
    Common:contains
    [ a RSLogix:Tag;                    # která obsahuje proměnnou
      RSLogix:name ?tagName;            # se stejným jménem
] ] ] }
```

Vidíme, že klást dotazy, jež jsme ve výkladu uvedli, je pomocí jazyka SPARQL možné relativně jednoduše – pro uživatele, který tento jazyk umí a který si pamatuje jména všech tříd a vlastností a zná jejich možné vztahy. Odměnou pro takovéhoto uživatele jsou bohaté možnosti dotazování. I v případě, že konstrukce dotazu zabere několik minut, je to pořád řádově méně, než by uživatel mohl strávit manuálním procházením všech dostupných projektů.

5.2.2.4 Uživatelské rozhraní pro konstrukci dotazů

Aby byla aplikace RoSS použitelná i pro uživatele, který se nechce učit speciální dotazovací jazyk, je třeba podstatně zjednodušit konstrukci dotazů. Charakter dat je v tomto případě blízký objektovému popisu – objekty jsou zařazené do tříd, objekty mají vlastnosti s hodnotami v podobě literálů různých typů a mají mezi sebou vazbu „obsahuje". Toho můžeme v naší aplikaci využít pro grafickou podobu konstrukce dotazu.



Obr. 5.4 Dotaz: „Najdi instrukci SUB, kde je jako operand použita konstanta 0,5".

Dotaz „Najdi instrukci SUB, kde je jako operand použita konstanta 0,5" pak vypadá tak, jak je ilustrováno na obr. 5.4. Vazba „obsahuje" je interpretována umístěním shora dolů. První obdélník shora říká, že hledáme objekt typu soubor (File) s vlastností jméno (name) a že toto jméno může být libovolné a jeho hodnota nás zajímá – a tedy je pro tuto hodnotu zavedena proměnná. Nejspodnější obdélník říká, že hledáme objekt typu operand (Operand) s konkrétní udanou hodnotou (value) „0,5". Tuto grafickou podobu dotazu uživatel zkonstruuje, uživatelské rozhraní pak daný dotaz přeloží do jazyka SPARQL, dotaz se vyhodnotí v bázi dat a uživatel dostane zpět výsledek v podobě možných kombinací hodnot zadaných proměnných.

Tato grafická podoba je přehlednější než textový zápis v jazyce SPARQL. Hlavní výhodou je ale interaktivní postup při tvorbě dotazu. Ukažme si postup tvorby dotazu na příkladu „Najdi rutiny, které nějak používají proměnnou, kterou zobrazuje libovolný objekt typu měřidlo". Pro ilustraci více možností si dotaz rozšíříme tak, že nás bude navíc zajímat i datový typ proměnné.

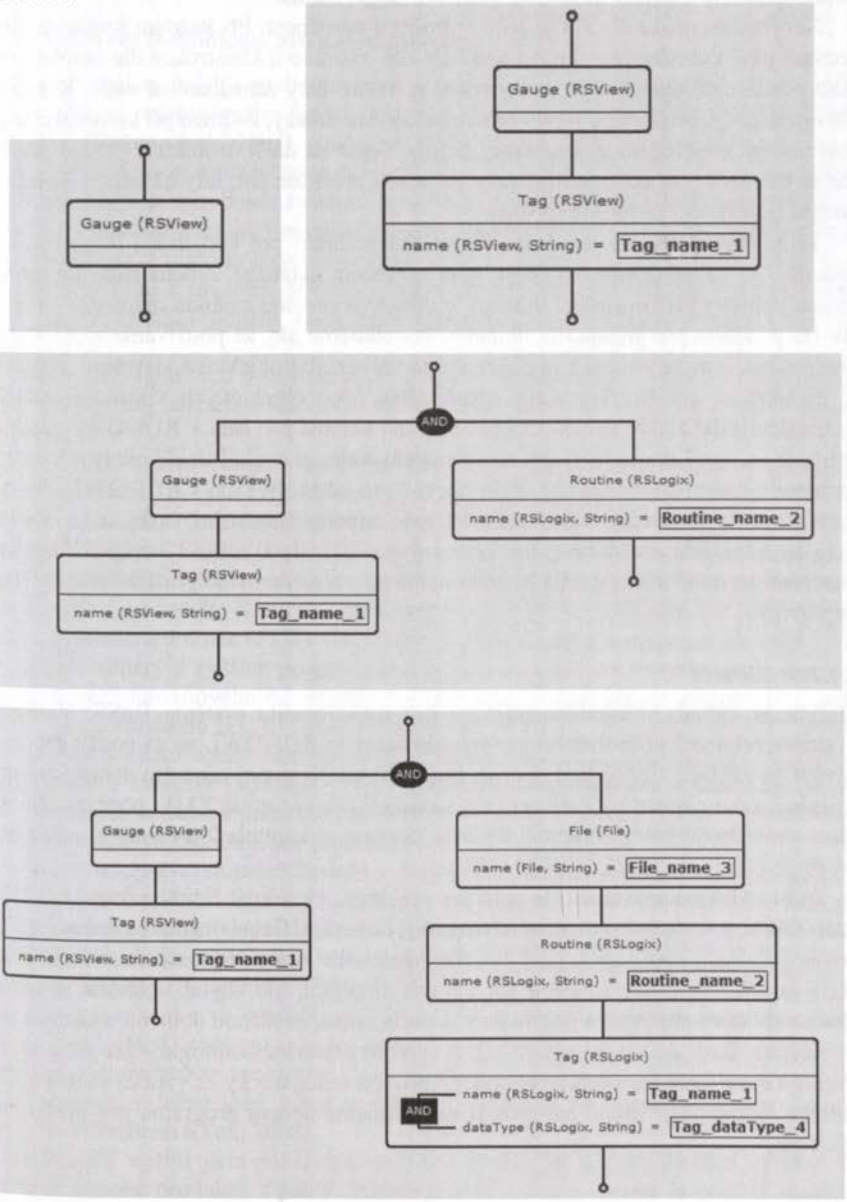
Dotaz začneme tak, že založíme nový element a z nabídnutého seznamu vybereme typ objektu, který nás zajímá – měřidlo (Gauge) jakožto grafický objekt na obrazovce RSView, viz obr. 5.5. Důležité je, že jsme typ objektu vybrali ze strukturované nabídky a není tedy třeba si název pamatovat nebo vypisovat, tak jako tomu bylo při použití jazyka SPARQL. Všimněme si též, že v tomto případě nás zajímá jen samotný objekt a žádné jeho vlastnosti. V dalším kroku přidáme proměnnou (Tag), u které nás zajímá její jméno, abychom ho mohli provázat s podobným objektem z žebříčkového kódu. Nyní přidáme druhou větev, pro žebříčkový kód, kde hledáme rutinu (Routine). Větev rozšíříme nahoře o soubor (File) a jeho jméno (name) a dole o proměnnou (Tag), tentokrát z domény žebříčkového kódu. Všimněme si, že pro atribut jméno (name) je použita stejná proměnná v obou větvích, a tím se zajistí provázání. Pro ilustraci je ukázáno, jak použít v rámci jednoho objektu více atributů – zde nás navíc zajímá datový typ.

Uživatelské rozhraní pro konstrukci dotazů v systému RoSS umožňuje libovolné větvení v rámci relace „obsahuje" mezi objekty stejně jako mezi atributy v rámci jednoho objektu, obojí s logickými vztahy AND i OR. Hodnotami atributů mohou být proměnné nebo konkrétní údaje a kromě vztahu „rovná se" lze při konstrukci dotazu využít i vztahy typu „menší než", „větší nebo rovno" apod.

Hlavní síla tohoto prostředí ale tkví v nabízení všech dostupných hodnot při konstrukci dotazu. Prostředí nabízí dva módy – dle ontologie a dle dat. První mód, dle ontologie, nabízí jen ty hodnoty, které jsou v daném místě vhodné. Pokud tedy vybíráme atribut pro třídu, dostaneme v nabídce jen ty atributy (datové vlastnosti v OWL), které mají danou třídu v definičním oboru. Pokud přidáváme nový objekt kamkoliv do dotazu, v nabídce dostaneme pouze takové třídy, které mohou v daném místě být dle relace „obsahuje" (objektové vlastnosti v OWL). Často je nabídka již po několika krocích natolik zúžená, že se nemusí dále strukturovat, výpis celého výběru se vejde na část obrazovky a konstrukce dotazu je tak velmi rychlá.

Druhý mód, dle dat, jde ještě dále a poskytuje mnohem větší pohodlí. Nabídka se řídí dle všeho, co je aktuálně uloženo v bázi dat. Již v prvním kroku tak dostaneme v nabídce pouze třídy, pro které existují nějaké objekty. Pro konkrétní hodnotu atributu nemusíme tuto hodnotu vypisovat, neboť dostaneme na výběr z toho, co je doopravdy v bázi dat, a to nikoliv obecně pro daný atribut, ale jen to, co je v daném kontextu celého dotazu možné. Pokud tedy například z ostatních omezujících podmínek dotazu vyplývá, že v daném místě nemůže být instrukce SUB, nedostaneme ji v nabídce pro konstrukci tohoto dotazu, ačkoliv se jinak v bázi dat vyskytuje. V tomto módu se tak mísí kon-

strukce dotazu a jeho odpovídání – při konstrukci dotazu tak jde spíše o prozkoumávání dat. Lze tak interaktivně přidávat a ubírat podmínky na základě aktuálního stavu znalostní báze.



Obr. 5.5 Kroky konstrukce dotazu „Najdi rutiny, které nějak používají proměnnou, kterou zobrazuje libovolný objekt typu měřidlo".

Dotaz, který takto zkonstruujeme, nemůže vrátit prázdný výsledek. Při konstrukci dotazu jen dle ontologie to samozřejmě možné je – v aktuální bázi dat nejsou vlastně nikdy obsazeny všechny možnosti, které ontologie nabízí.

Nevýhodou módu dle dat je jeho výpočetní náročnost. Při každém kroku se vlastně provádí plné vyhodnocení dotazu nad bází dat, zatímco u konstrukce dle ontologie nám stačí pouze ontologie – ta je v porovnání se všemi daty zanedbatelně malá. Konstrukce dle ontologie je použitelná vždy i pro velmi složité dotazy, zatímco při konstrukci dle dat můžeme při složitějším dotazu velmi dlouho čekat na další nabídku k výběru. Mód dle dat se tak hodí pro jednodušší dotazy nebo pro průzkum dat, kdy až během konstrukce dotazu zjišťujeme, co přesně hledáme.

Všimněme si též, že uvedené uživatelské rozhraní pro konstrukci dotazu je velmi obecné – je řízeno pouze ontologií, popř. obsahem databáze. Změna rozhraní (přesněji řečeno nabídky při konstrukci dotazu) je dosažena prostou změnou ontologie – rozhraní tak lze používat pro jakoukoliv doménu. Podotkněme ale, že používáme zjednodušující předpoklad, a to že v datech je pouze jedna univerzální objektová vlastnost „obsahuje". V doméně, v níž RoSS pracuje, nepřinášelo více objektových vlastností vzhledem k charakteru dat žádný užitek. Obecně toto ale nemusí pro data v RDF/OWL platit, pro aplikace na méně kontrolovaném sémantickém webu to neplatí téměř nikdy. Uživatelské rozhraní tak není v této podobě zcela obecné pro jakákoliv data v RDF/OWL. Pro ontologie s více objektovými vlastnostmi by bylo zapotřebí rozšířit i vazby mezi objekty – bylo by zapotřebí dle definičního oboru a oboru hodnot nabízet konkrétní objektové vlastnosti na dané místo, a dále by bylo nutné takovéto vazby do grafické podoby dotazu vypisovat.

5.2.2.5 Shrnutí

Shrňme si výhody uvedeného přístupu pro funkcionalitu systému RoSS. Vidíme, že výsledek relativně přímočarého převodu do formátu RDF/OWL se dá použít pro vyhledávání na základě složitějších dotazů. Jazyk SPARQL je pro takovéto dotazování určitě užitečný. Jako alternativy k uložení dat se nabízí přímo formát XML, popř. relační databáze – nevýhody tohoto přístupu již byly uvedeny v kapitole 2 a čtenář si může ověřit jejich platnost i pro tuto konkrétní aplikaci.

Další výhodou je řízení celé aplikace ontologií. Ontologií řídíme převod z XML do RDF/OWL, a to včetně možnosti odvozování důsledků. Celá konstrukce dotazů je řízena pouze ontologií, popř. celou bází dat. Pro jakoukoliv změnu formátu dat, včetně přidání zcela nového typu dat, je třeba jen upravit ontologii, ale zbytek aplikace se nemění. Změna aplikace pro práci s jinými daty, i zcela mimo uvedenou doménu, znamená zajistit možnost čtení dat ve formátu XML a vytvořit příslušné ontologie – jak jsme již však uvedli, i tento krok lze v základní podobě provést automaticky ze vzorku souborů XML. Údržba ontologie je určitě jednodušší než případná úprava programu pro přidávání jiných dat.

5.3 Sémantika v holonických a multiagentních průmyslových řídicích systémech

5.3.1 Agentové technologie pro automatizaci

Technologie holonických a multiagentních systémů se ve stále větší míře začínají uplatňovat v oblasti průmyslového řízení. Jak již bylo nastíněno ve čtvrté knize o Umělé inteligenci (Mařík a kol., 2003), současné trendy v oblasti automatizace ukazují na odklon od tradičních centralizovaných přístupů směrem k přístupům distribuovaným. Stále vzrůstající složitost výrobních systému a postupů, požadavky na schopnost zvládat nepředvídané poruchy a stavy, flexibilita a rekonfigurovatelnost výrobního systému požadovaná zejména u malosériových výrob se širokou paletou výrobků, časté modifikace výrobních postupů a zavádění nových výrobků, to vše jsou faktory, které značně komplikují, ne-li vylučují použití klasických hierarchických a centralizovaných řídicích struktur.

Teorie multiagentních a holonických systémů, vycházející z oblasti distribuované umělé inteligence, nacházejí v současnosti uplatnění na všech úrovních řízení a organizace výroby.

Na nejnižší úrovni **řízení v reálném čase** je zřejmá přímá interakce mezi agenty a jimi kontrolovaným fyzickým zařízením. Vzhledem k nutnosti zaručit odezvu řídicího systému v reálném čase byla navržena architektura nazývaná *holon* (Van Leuven, Norrie, 1997), *fyzický agent* (Fipa, 2002) nebo *holonický agent* (Vrba, 2006). Ta obsahuje vedle samotného softwarového agenta, pro inteligentní rozhodování na vyšší úrovni, i modul nízkoúrovňového řízení v podobě programu běžícího na klasickém programovatelném logickém automatu (PLC). Na rozdíl od agenta, jehož komplexní chování vyžaduje použít některý z vyšších programovacích jazyků, jako je C++ nebo Java, jsou pro subsystém nízkoúrovňového řízení stále využívány jazyky pro programování PLC, specifikované ve standardu IEC 61131-3. Postupně se začíná využívat i standard IEC 61499 tzv. funkčních bloků, který více vyhovuje distribuovanému a modulárnímu charakteru multiagentních systémů (Vyatkin, 2007). Interakce mezi agentem a nízkoúrovňovým řízením probíhá nejčastěji prostřednictvím přímého čtení a zápisu hodnot v paměti PLC (tzv. tagů v datové tabulce), jež jsou běžně využívány programy běžícími v PLC pro uchovávání vstupně-výstupních hodnot a dalších pomocných řídicích veličin. Někdy je za součást zmíněné holonické architektury považováno i samotné fyzické zařízení (Vyatkin a kol., 2005), (Lepuschitz a kol., 2009). Příkladem úspěšného nasazení je například výrobní linka pro montáž hlav válců instalovaná v závodě Daimler Chrysler ve Stuttgartu, kde použití agentové technologie prokázalo zvýšenou odolnost řídicího systému vůči poruchám a ve výsledku o 20 % větší propustnost výrobní linky (Bussman, Schild, 2001). Další úspěšnou aplikací je multiagentní řídicí systém vyvinutý společností Rockwell Automation ve spolupráci s americkým námořnictvem určený pro řízení chladicího systému válečné lodě, který je schopný fungovat i při poškození nebo zničení svých částí (Brennan a kol., 2008).

Inteligentní agenti jsou používáni také pro **plánování a rozvrhování** výroby. Na rozdíl od masové produkce s plány fixními po celou dobu výroby je použití agentové technologie výhodné při plánování tzv. projektově orientované výroby, kde jsou jednotlivé výrobky víceméně unikátní, a tedy i výrobní plán je jedinečný pro každý výrobek.

Systém ExPlanTech vyvinutý Gerstnerovou laboratoří na ČVUT (Pěchouček a kol., 2005) modeluje individuální výrobní jednotky, jako jsou výrobní oddělení, úseky a stroje, pomocí agentů, které mezi sebou vyjednávají alokaci zdrojů pro výrobky s rozdílnými prioritami a daty požadovaného dokončení. V současnosti je systém nasazen a běžně používán pro plánování výroby v Modelárně LIAZ, s. r. o., která patří k předním evropským výrobcům forem, odlitků a výlisků. Koncept multiagentního plánování vyvinutý v rámci projektu ExPlanTech byl aplikován také při vývoji plánovacího systému pro závod společnosti ŠkodaAuto, a. s., na montáž motorů, jehož produkce dosahuje tisíců kusů a vyznačuje se velkou variabilitou vyráběných motorů (Pěchouček a kol., 2007).

Související oblastí, kde dochází k úspěšnému komerčnímu nasazení agentů, je **logistika**. Společnost Magenta Technology vyvinula multiagentní plánovací nástroj i-Scheduler založený na konceptu virtuálního trhu, kde jednotliví agenti, kteří reprezentující například řidiče, vozidla, nákladní depa, cílové stanice apod., sdílejí informace a vyjednávají o způsobu, času a ceně dopravy (Rzevski a kol., 2007). Systém je nasazen například u velkého britského kamionového přepravce s flotilou čítající stovky vozidel a s více než tisícovkou vyřizovaných objednávek denně. Další aplikací je plánování mezinárodní přepravy ropy tankery společnosti Tankers International (Skobelev, 2011) nebo též plánování dodávek paliva, materiálu, potravin a vody na mezinárodní vesmírnou stanici ISS (Andreev a kol., 2007).

Na nejvyšší úrovni poskytují agentové technologie vhodnou platformu pro realizaci vize tzv. **virtuálních podniků** (Virtual Enterprises) nebo **virtuálních organizací** (Virtual Organization). Virtuálním podnikem se rozumí dočasná aliance podniků, která je vytvořena za účelem sdílení schopností a zdrojů k získání výhodnější pozice v obchodním prostředí (Camarinha-Matos, 2002). Kooperace v rámci virtuálního podniku je založena na výměně informací prostřednictvím počítačových sítí. Virtuální podnik navenek vystupuje jako jediná entita bez toho, aby bylo nutné vytvářet nový právní subjekt či zřizovat fyzické sídlo.

Hlavní charakteristikou jakéhokoliv multiagentního systému je inteligentní zpracování znalostí a vzájemná komunikace mezi agenty při společném řešení úloh. Pro efektivní zpracování a výměnu informací je potřeba, aby informace byly srozumitelné pro všechny agenty a také měly stejný, jednoznačně definovaný, význam neboli sémantiku. Je tedy vhodné, aby sémantika byla popsána v explicitní a automaticky zpracovatelné podobě, tj. pomocí ontologie dostupné všem agentům, a aby báze znalostí agentů stejně jako informace obsažené ve zprávách zasílaných mezi agenty byly s danou ontologií kompatibilní.

Výzkum v oblasti multiagentních řídicích systémů začíná v poslední době zaměřovat svoji pozornost právě na možnosti aplikace sémantiky a ontologií. Ukazuje se, že pouhá syntaktická interoperabilita, vesměs zajištěná použitím jazyka XML (Extensible Markup Language) a standardy pro agentní komunikaci, poskytovanými například organizací FIPA (Fipa, 2002), je nedostatečná. Nástup nových internetových technologií, jako je sémantický web (Semantic Web) a sémantické webové služby (Semantic Web Services), motivoval, a v posledních několika letech významně urychlil, aplikaci sémantických technologií v oblasti multiagentních průmyslových řídicích systémů.

5.3.2 Doménově specifické ontologie

Ontologie navrhované pro oblast průmyslové výroby se dají rozdělit z obecného pohledu do dvou skupin:

a) doménově specifické ontologie, o nichž je diskutováno v této kapitole a které pokrývají pouze úzkou, specifickou oblast výroby a jsou navržené většinou pouze pro účely konkrétní (agentní) řídicí aplikace a

b) obecné ontologie navržené s důrazem na integraci existujících standardů a norem a jejich převedení do automaticky zpracovatelné podoby.

Společným znakem ontologií navržených pro konkrétní multiagentní řídicí aplikace je snaha vytvořit formální model pro obecný popis kategorií výrobních komponent, jejich parametrů a poskytovaných operací. Pomocí ontologie také popisujeme výrobní postupy pro jednotlivé typy výrobků, nejčastěji v podobě sekvence výrobních operací s příslušnými parametry. Důležitým prvkem je, že tento popis je obecný, nezávislý na konkrétním uspořádání výrobního závodu. Plánování výroby, tj. přidělování operací z výrobního postupu jednotlivým strojům se provádí dynamicky, kdy se pro každou operaci vyhledají a kontaktují v tu chvíli dostupné stroje, resp. agenti, kteří je reprezentují, s žádostí o určení ceny, za kterou jsou schopny danou operaci poskytnout, popř. nejbližšího termínu, kdy mohou operaci provést. Stroji nabízejícímu nejnižší cenu či nejbližší termín je následně potvrzena žádost o rezervaci jeho kapacity pro daný výrobek a čas a ostatním strojům jsou jejich nabídky zamítnuty. Poté je zahájen transport výrobku od stroje, který provedl předchozí operaci, ke stroji nově zvolenému.

Jedním z pracovišť, kde se zabývají využitím ontologií v multiagentních řídicích systémech, je Intelligent Robotic Center na univerzitě Uninova v Portugalsku (Cândido a kol., 2007). Laboratorní model montážní linky obsahuje dva roboty, každý se čtyřmi různými druhy úchytů a nástrojů, a automatický dopravní systém propojující roboty a sklad. Pro popis linky byla vytvořena ontologie obsahující dva základní koncepty: `modul`, reprezentující výrobní jednotku či jejich agregaci, a `schopnost` s významem schopnosti provádět výrobní operaci. Příkladem modulu je `pracovní buňka` (workcell) skládající se z `pracovních stanic` (workstation), které se dále dělí na `jednotky` (unit). Jednotkou je například `transformační jednotka` (fréza, robot apod.), `dopravní jednotka` (pás) a `verifikační jednotka`. Pro zachycení kompozice prvků ontologie, resp. dědičnosti v hierarchii, podobně jako v objektově orientovaných jazycích, se používá dvou v ontologiích typických konstrukcí `skládá se z` („composed of"), popř. `je` („is-a") s významem „je podtřídou třídy". Stejně jako jednotky se i schopnosti dále specializují, jako např. `lineární pohyb` („move linear"), přičemž komplexní schopnosti lze reprezentovat jako agregaci schopností jednoduchých. Základním elementem v multiagentním řídicím systému je agent výrobního zdroje, který danou ontologii používá pro reprezentaci znalostí o výrobním systému a k uvažování o objektech a jejich vztazích. Tento agent, reprezentující například robota, po svém vytvoření prohledá ontologii, kde podle svého sériového čísla a vybavení získá seznam svých schopností, které následně zaregistruje v tzv. zlatých stránkách multiagentního systému. Službu zlatých stránek (v angličtině se používá termín yellow pages) zajišťuje dedikovaný agent, Directory Facilitator (DF), který spravuje informaci o službách poskytovaných agenty a umožňuje jiným agentům vyhledat vhodného poskytovatele požadované služby. Pro poskytnutí komplexních operací tvoří jednotliví agenti výrobních zdrojů koalice, jež

jsou vedeny lídrem koalice, který zaregistruje komplexní službu ve zlatých stránkách a pak koordinuje její provádění členy koalice. Podle slov autorů přispělo navržené řešení k mnohem snazší rekonfigurovatelnosti výrobního systému. Komponenty mohou být do systému přidávány, nebo z něj odebírány, v podstatě za chodu, bez nutnosti složitě rekonfigurovat řídicí systém, a tak se může linka snadno adaptovat na nové typy produktů a nové výrobní postupy.

Další příklad ontologie integrované v multiagentním systému určeném pro řízení malé laboratorní výrobní linky je uveden v práci (Al-Safi, Vyatkin, 2007). Linka se skládá ze dvou strojů vybavených různými nástroji jako razníkem, vrtačkou, rotační indexovací stolicí atd. Základními koncepty ontologie jsou `zdroj` (resource) a `operace` (operation). Při porovnání s předchozím příkladem je sice použito odlišné názvosloví (`modul` versus `zdroj` a `schopnost` versus `operace`), nicméně význam je v obou případech totožný. Zdroje se dále specializují na `stroj`, popř. `nástroj`, s podtřídami `manipulační stroj` (např. `rotační indexovací stolice`) a `výrobní stroj` (např. `vrtačka`). Operace jsou rozděleny na `výrobní operace`, popř. `logistické operace`, s další specializací na `třídění`, `vrtání`, `testování otvoru`, popř. `vyjmutí`, `rotace` atd. Důležitým prvkem v každé ontologii jsou vazby mezi koncepty. Zde je to vazba `umožňuje realizaci` (enables realization of) mezi koncepty `zdroj` a `operace`, pomocí níž je vyjádřena například vazba „výrobní stroj umožňuje realizaci operace vrtání". Tyto obecné koncepty a vazby ontologie jsou pak použity pro zachycení reálného stavu konkrétního výrobního systému ve znalostních bázích agentů řídících výrobní proces. Formální popis systému pomocí ontologie (modelované v jazyce OWL) umožňuje agentům efektivně rozhodovat o plánování výroby, tj. výběru vhodných strojů poskytujících požadované operace.

V Institutu automatizace a řízení na Technické univerzitě ve Vídni byla vytvořena ontologie zaměřená na problematiku přepravy materiálu ve výrobním závodě (Merdan a kol., 2008). Ontologie je používána agenty reprezentujícími základní prvky dopravního systému, jako je dopravní pás, výhybka, dokovací stanice, paleta apod., k zachycení reálné situace v dopravním systému, tj. vzájemného propojení komponent, aktuální pozice palet, poruchových stavů atd. Tyto informace jsou v sémantické podobě uloženy ve znalostních bázích agentů, které jsou průběžně aktualizovány tak, jak každý agent vnímá okolní prostředí pomocí svých senzorů, nebo přijímá informace ve zprávách zasílaných okolními agenty. Každá změna ve znalostní bázi, např. informace o sepnutí senzoru nebo poruše sousední komponenty, aktivuje odpovídající chování agenta a vyvolá tak reakci na nově získanou informaci. Zmíněná ontologie byla rozšířena i pro účely plánování a rozvrhování výroby. `Produkt` je zde reprezentován jako kompozice `dílčích sestav` (subassembly), které dále obsahují `části` (part) představující `surový materiál` (raw material). Každá dílčí sestava je výsledkem výrobního `kroku`, ve kterém je určitá `operace` provedena `výrobním zdrojem`. Postupné provádění kroků výroby je řízeno plánovacími agenty v součinnosti s agenty výrobních zdrojů, a to v pořadí, které je ve výrobním plánu vyjádřeno pomocí vazeb `vyžaduje předchůdce` a `je následován` mezi jednotlivými kroky výroby.

Pro podporu organizace znalostí v dodavatelsko-odběratelských řetězcích v automobilovém průmyslu byla vytvořena ontologie FRISCO (Hellingrath a kol., 2009). Ontologie pokrývá širokou škálu informací týkajících se produktů, součástek, výrobních zdrojů, plánování výroby, časových a datových údajů apod., které jsou předávány mezi výrob-

ními závody, prodejci a zákazníky. Aplikovatelnost ontologie byla ověřována na agentním modelu zahrnujícím jeden závod na výrobu automobilů a dva dodavatele součástek, reprezentované agenty, kteří ontologii používají pro komunikaci týkající se dodávek dílů.

Společností, která je zřejmě nejdále v reálném komerčním nasazení ontologií v kombinaci s agenty, je již zmíněná Magenta Technology. Ta vyvinula ontologii pro oblast logistiky, jež je využívána pro vyjednávání o času a ceně dopravy mezi agenty reprezentujícími řidiče, vozidla, nákladní depa, cílové stanice apod. (Rzevski a kol., 2007).

5.3.3 Obecné ontologie

U obecných ontologií je důraz kladen na integraci s existujícími systémy. Často jde o popis existujících standardů ve formě vhodné pro automatizované zpracování. Jako příklad lze uvést popis funkčních bloků, událostí, hardwarového a softwarového vybavení apod. Podkladem pro tyto ontologie jsou normy, například ANSI/ISA 95 a 88 (ANSI/ISA, 1995). Norma ISA-95 „Enterprise-Control System Integration" popisuje hierarchický model výrobní organizace, tok událostí atd. a slouží tak jako základ pro integraci řídicího systému s obchodními systémy podniku. Norma ISA-88 „Batch Control" popisuje ve větších detailech prostředí procesů dávkového zpracování. Definuje zejména hierarchický model výrobního systému od podniku přes oblasti a jednotky až k řídicím modulům. Dále definuje procesní model pro popis uspořádaných výrobních etap a výrobních akcí.

O standardizaci v oblasti sémantiky pro automatizaci se snaží například organizace O3NEIDA (zkratka anglického názvu Open Object-Oriented kNowledge Economy for Intelligent inDustrial Automation). Jejím cílem je vytvořit otevřenou technologickou infrastrukturu pro automatizační komponenty (Vyatkin a kol., 2005) k zajištění softwarové a hardwarové kompatibility na všech úrovních trhu s automatizačními komponentami, tj. od výrobců součástek, zařízení a strojů, přes systémové integrátory až po výrobní podniky. Základním prvkem architektury je tzv. *automatizační objekt*, který je abstrakcí mechanického zařízení se zapouzdřenou inteligencí, tj. softwarovou komponentou, jež poskytuje různé funkce, jakými jsou řízení, vizualizace, simulace, diagnostika atd. Jednoduché automatizační objekty, jako jsou senzory, pohony motorů nebo mikroprocesory, představují znovupoužitelné moduly, z nichž je možné skládat komplexní automatizační objekty (např. stroje) a z těch stejným modulárním způsobem celé výrobní podniky. Sémantický popis automatizačních objektů pomocí technologií sémantického webu je prezentován v práci (Lopez, Lastra, 2006). Navržená ontologie pro automatizační objekty, která byla navržena ke standardizaci skupině IEC-TC65, sdružuje referenční model mechatronických zařízení a referenční model funkčních bloků standardu IEC 61499 pro holonické řízení v reálném čase.

Standardizaci v oblasti automatizace věnuje nemalé úsilí též americký národní úřad pro standardy a technologie (NIST). Příkladem je datový model výrobního závodu (shop data model) vyjádřený pomocí jazyků UML (Unified Modeling Language) a XML (Extensible Markup Language), viz (McLean a kol., 2005). Model obsahuje popis organizace, výrobních plánů, zdrojů, rozvrhů apod., nicméně není formální ontologií, jak ji chápeme v kontextu této kapitoly. Druhým příkladem aktivit úřadu NIST je jazyk PSL (Process Specification Language), který představuje logickou teorii pokrývající obecnou reprezentaci procesů v průmyslových aplikacích (Grüninger, Kopena, 2005). Ontologie

PSL obsahuje obecné koncepty, jako jsou například `aktivita`, `objekt`, `časový bod`, funkce, jakými mohou být například `začátek` nebo `konec`, a vazby jako například `mezi`, `odehrává se v`, které jsou sdruženy do různých oblastí, např. pro popis komplexních aktivit.

Dalším příspěvkem ke standardizaci je ontologie MASON (MAnufacturing's Semantic ONtology) prezentovaná v sborníku (Lemaignan a kol., 2006). Cílem jejích autorů je vytvořit obecnou ontologii, která by umožňovala hladkou integraci dalších, doménově specifických ontologií zavedením jednotné kognitivní architektury. Ontologie je vytvořena v jazyce OWL a zavádí taxonomii konceptů, jakými jsou entita, operace, zdroj, a jejich vazeb. Byl též vytvořen převodní nástroj mezi touto ontologií a interním ontologickým modelem používaným v populární agentní platformě JADE (http://jade.tilab.com). Ačkoliv se některé konstrukce ontologie zdají být aplikačně specifické a zbytečně svazující, je tato práce považována za důležitý krok vpřed k formalizaci sémantiky pro oblast automatizace.

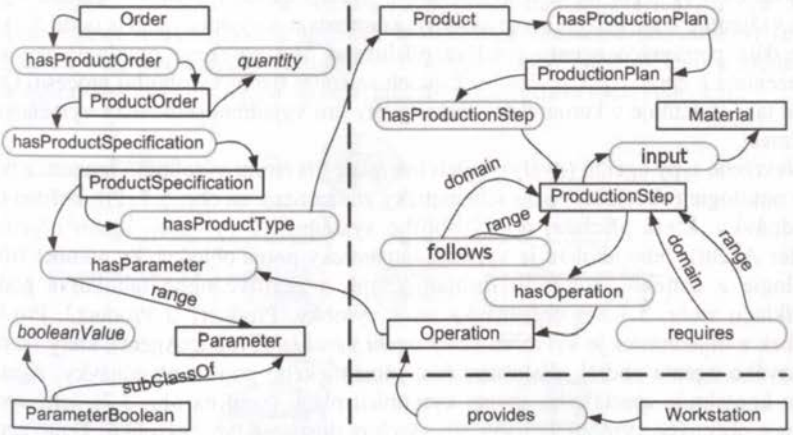
5.4 Návrh obecné ontologie a multiagentní architektury pro automatizaci

Z předchozích dvou odstavců 5.3.2 a 5.3.3 je patrné, že existuje celá řada ať již obecných nebo aplikačně specifických ontologií pro různé oblasti průmyslové výroby. V tomto oddílu je zachycena snaha firmy Rockwell Automation vytvořit nové obecné ontologie zaměřené na oblast diskrétní automatizace a současně vytvořit obecné architektury distribuovaného, multiagentního řídicího systému, který přirozeně integruje ontologii pro sémantický popis a zpracování znalostí o výrobních procesech, zařízeních a produktech. Hlavní důraz byl kladen na aplikovatelnost a možnost reálného nasazení v průmyslové praxi, a to v širokém spektru úloh. Z toho vyplývají dva vzájemně související atributy. Zaprvé, ontologie musí být dostatečně obecná, tj. poskytovat společný konzistentní model základních konceptů, na nichž pak budou stavěny aplikačně specifické rozšiřující ontologie pro různé domény. Zadruhé, ontologie musí být relativně jednoduchá, proto aby byla aplikovatelná v reálném provozu. Obecným problémem existujících ontologií je totiž jejich přehnaná složitost, která se odráží v neúměrných výpočetních a paměťových nárocích při počítačovém zpracování, zejména při automatickém uvažování. To může představovat zásadní problém především pro nejnižší úrovně řídicího systému, kde řídicí programy, které by měly pracovat s ontologiemi, běží na vestavěných zařízeních či automatech PLC s omezenou výpočetní silou.

5.4.1 Ontologie

Ontologie vytvořená v jazyce OWL zahrnuje tři různá hlediska diskrétní automatizace:
– specifikaci zákaznické objednávky,
– definici výrobního plánu a
– popis struktury výrobního a transportního systému.
Schématické znázornění ontologie je ve zjednodušené podobě, tj. pouze s vybranými prvky, zachyceno na obr. 5.6. Koncepty (třídy) ontologie jsou znázorněny obdélníky, obdélníky se zaoblením pak vyjadřují vazby mezi koncepty ontologie.

Objednávka (`Order`) obsahuje seznam objednávaných položek (`ProductOrder`), kterých může být 1 až *N*. Každá z položek obsahuje specifikaci výrobku (`ProductSpecification`), tj. zejména typ objednávaného výrobku (vazba `hasProductType`), jeho parametry (vazba `hasParameter`) a objednávané množství (vazba `quantity`). Jedna objednávka tak může například obsahovat požadavek na deset kusů výrobku typu A a dvacet kusů výrobku typu B a pro každý typ obsahovat seznam specifických parametrů, jako je třeba barva, použitý materiál, rozměry apod. K popisu parametrů slouží obecná třída `Parameter`, od níž lze snadno odvozovat specifické typy parametrů, jako například v obr. 5.6 zachycený `ParameterBoolean`, pomocí nějž je možné vyjádřit logickou hodnotu `true` nebo `false`.



Obr. 5.6 Segment ontologie pro diskrétní automatizaci – levá část se týká specifikace objednávky, pravá horní část představuje obecný popis výrobního postupu výrobku a pravá dolní část pak slouží pro popis struktury výrobního systému.

Segment ontologie týkající se objednávek (levá část obr. 5.6) je propojen přes vazbu `hasProductType` s typem požadovaného výrobku (`Product`). Stejně jako u parametrů je i zde možné od obecné třídy prokuktu odvozovat pro daný výrobní závod konkrétní typy výrobků. Každý typ výrobku má přiřazen přes vazbu `hasProductionPlan` svůj plán výroby (`ProductionPlan`). Ten sestává ze série diskrétních výrobních kroků (`ProductionStep`), jejichž pořadí provádění lze definovat pomocí vazeb `follows` s významem „krok B následuje po kroku A" (nejde o bezprostřední následnost, tj. mezi kroky A a B mohou být vykonány další kroky) a `immediatellyFollows` s významem „krok B následuje bezprostředně po kroku A" (mezi A a B nemohou být vloženy jiné kroky). Kombinací těchto vazeb lze vytvářet sériové nebo paralelní větve kroků a jejich libovolné kombinace. V každém kroku je na výrobku provedena určitá operace (`Operation`) s příslušnými parametry (viz vazba `hasParameter`) a materiálem (třída `Material` a s vazbou `input`).

Operace jsou ve výrobním systému poskytovány výrobními stanicemi (`Workstation`). Výrobní stanice může představovat samostatný stroj nebo seskupení více strojů

a zařízení. V druhém případě je obvykle výrobní operace uvnitř výrobní stanice dekomponována na sérii elementárních operací prováděných jednotlivými zařízeními. Pro dopravu výrobků a materiálů mezi výrobními stanicemi zahrnuje tato část ontologie definici transportních uzlů propojených transportními hranami (není znázorněno na obr. 5.6). Výrobní stanici samotnou nebo některé z jejích zařízení je pak možné označit jako uzel v transportním systému. Hranami jsou například dopravní pásy nebo automatické samohybné vozíky AGV.

5.4.2 Integrace ontologie v distribuovaném řídicím systému

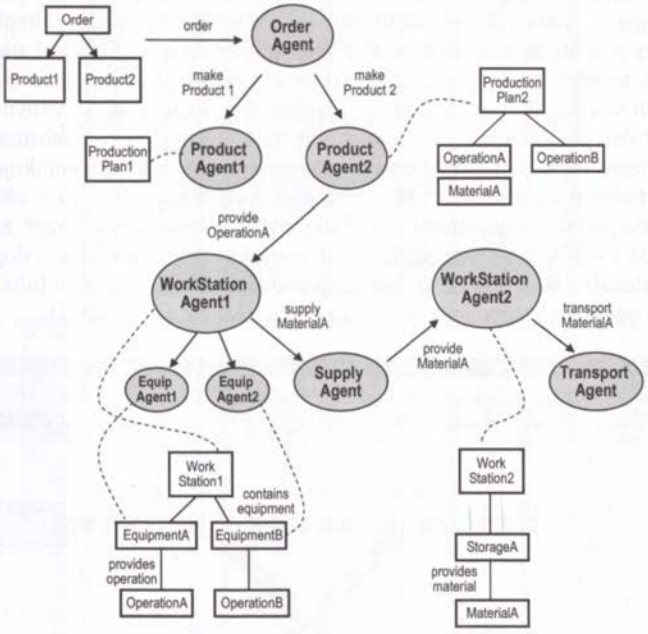
Ontologie představená v předchozím odstavci slouží jako základ fungování distribuovaného řídicího systému na bázi autonomních inteligentních jednotek – agentů – schopných vzájemně koordinovat svoje aktivity, kooperovat a vyjednávat (viz oddíl 5.3). Každý z dále popsaných agentů používá příslušnou část navržené ontologie pro vnitřní reprezentaci a zpracování znalostí týkajících se stavu řízení výrobního procesu. Ontologie se také uplatňuje v komunikaci mezi agenty pro vyjádření sémantiky vyměňovaných informací.

Navržené typy agentů (ovály), jejich interakce při řízení výrobního procesu a použité třídy ontologie (obdélníky) jsou schematicky znázorněny na obr. 5.7. Pro každou novou objednávku, která přichází do výrobního systému, je vytvořen *agent objednávky* (Order Agent). Jeho úkolem je vytvořit sémantický popis objednávky pomocí zmíněné ontologie z daného vstupního formátu (např. v textové nebo tabulkové podobě). V příkladu z obr. 5.7 zní objednávka na 2 výrobky, Product1 a Product2. Pro každý výrobek z objednávky je vytvořen nový *agent výrobku* (ProductAgent), který od objednávkového agenta obdrží příslušnou část sémantického popisu objednávky. Agent výrobku kontaktuje speciálního agenta výrobních plánů (není na obr. 5.7), jenž spravuje instance obecných výrobních plánů pro všechny dostupné typy výrobků. Tento agent na základě parametrů z objednávky modifikuje obecný plán a vytvoří tak plán šitý na míru konkrétní instanci objednávaného výrobku. Tím je myšleno zejména přenesení parametrů z objednávky (např. barva výrobku) do parametrů výrobního plánu (krok lakování), popř. vyjmutí některých kroků v případě, kdy příslušná vlastnost výrobku není v objednávce zaškrtnuta. Agent produktu tak obdrží specifickou instanci výrobního plánu, podle níž pak výrobu sám řídí (viz ProductionPlan2 jako plán výroby pro agenta ProductAgent2 na obr. 5.7).

Agent výrobku je klíčovým prvkem celého systému. Na rozdíl od obecného chápání produktu jako pasivní entity reprezentuje agent výrobku koncept aktivní, inteligentní entity svázané s fyzickým produktem, která uchovává informace o svém stavu a aktivně se podílí na plánování výrobních operací v součinnosti s ostatními agenty (Meyer a kol., 2009). Tento agent automaticky zpracovává plán výroby, a to sekvenčně, krok po kroku v pořadí daném výrobním postupem. Zpracování každého kroku probíhá ve třech fázích, vyjednáváním s agenty výrobních stanic. Pro toto vyjednávání byl vytvořen vlastní protokol Plan-Commit-Execute (Kadera, Tichý, 2009), který je rozšířením standardního vyjednávacího protokolu typu contract-net (Smith, 1980).

V *plánovací fázi* agent výrobku kontaktuje agenta tzv. zlatých stránek, což je standardní komponenta multiagentního systému zpravující informace o službách nabízených ostatními agenty. V tomto případě jde o *agenty výrobních stanic*, které mají zaregistrovány svoje výrobní operace. Agent výrobku tak pro daný krok získá seznam agentů

výrobních stanic nabízejících operaci spojenou s tímto krokem (viz např. OperationA ve výrobním plánu agenta ProductAgent2 na obr. 5.7). Následně pak agent výrobku osloví všechny agenty výrobních stanic ze seznamu s žádostí o nabídku naplánování operace. Agent výrobní stanice zkontroluje svůj kalendář, kde jsou již dříve sjednané operace, a vyhledá nejbližší vhodné časové okno, kdy by mohl požadovanou operaci provést. Při tom bere v úvahu dobu potřebnou pro vlastní operaci, dobu dopravy výrobku z jeho aktuální pozice a též třeba dobu dopravy materiálu, který je pro operaci potřeba a přitom není k dispozici přímo ve stanici. Tento výpočet je celkem komplexní problém, proto je pro něj dedikován speciální *agent dodávky* (SupplyAgent). V příkladu uvedeném na obr. 5.7 je pro operaci OperationA potřeba materiál MaterialA. Stejně jako operace je i materiál nabízen výrobními stanicemi. Agent dodávky tedy postupuje podle stejného scénáře, tj. osloví příslušné výrobní stanice s žádostí o dodávku materiálu (na obr. 5.7 je to výrobní stanice reprezentovaná agentem WorkStationAgent2). Doba dopravy se určí v součinnosti s *agenty dopravy* (TransportationAgent), které mají přehled o aktuálním stavu v dopravním systému a jsou schopni nalézt nejkratší trasu mezi stanicemi.



Obr. 5.7 Distribuovaný inteligentní řídicí systém na bázi agentů a ontologií.

V další fázi, nazývané *potvrzovací fáze*, vybere agent výrobku nejlepší nabídku, tj. nejbližší možný termín provedení operace a příslušnému agentu výrobní stanice pošle potvrzení, které je současně žádostí o zarezervování nabídnutého časového okna. Ostatním agentům výrobních stanic odešle zamítavou reakci. Vybraný agent výrobní stanice pak stejným způsobem požádá agenta dodávky o potvrzení rezervace materiálu a dopravy.

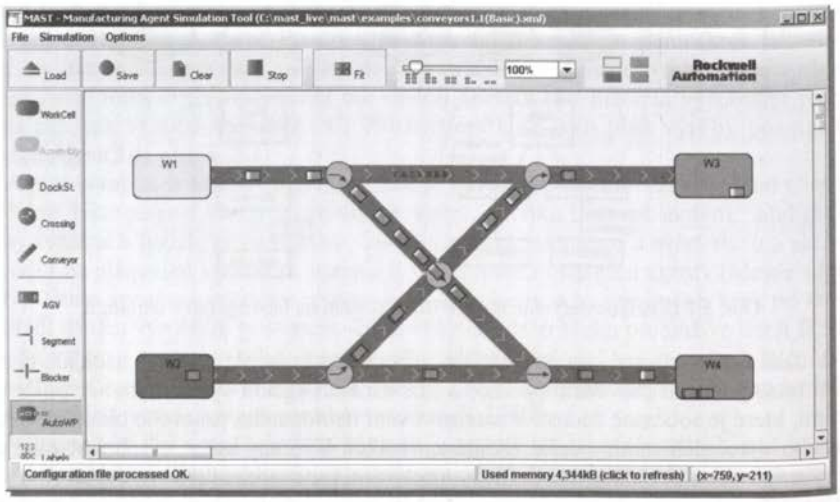
Ve *vykonávací fázi*, která je většinou zahájena po dokončení předchozího kroku výroby (plánovací a potvrzovací fáze mohou proběhnout s předstihem), pošle agent výrob-

ku žádost agentovi výrobní stanice o zahájení objednané operace. Ten nejprve požádá agenta dodávky o zahájení přepravy materiálu a samotného výrobku z jeho aktuální pozice. Po dokončení dopravy se spustí vlastní operace, jejíž provádění koordinuje agent výrobní stanice s podřízenými agenty zařízení. Po dokončení operace může agent výrobku přejít k dalšímu kroku ve výrobním plánu. Po skončení všech kroků výroby informuje agent výrobku objednávkového agenta o dokončení své práce.

5.5 Sémantické rozšíření agentního řídicího systému přepravy materiálu

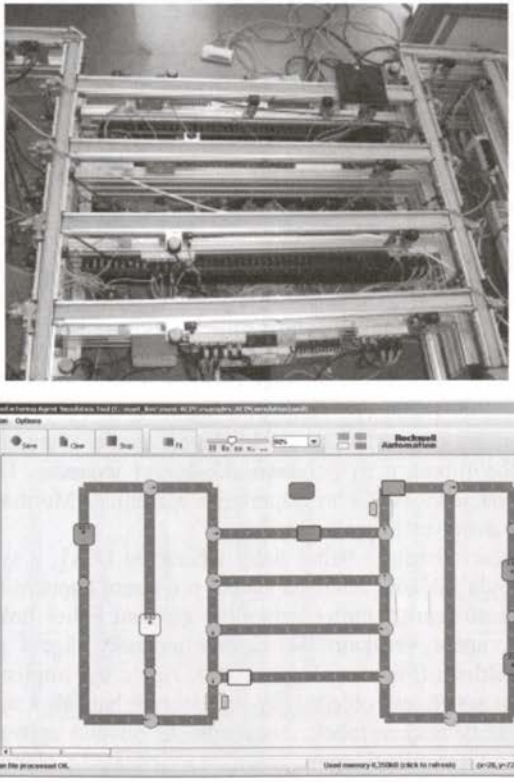
5.5.1 Systém MAST

Multiagentní simulační systém MAST (Manufacturing Agent Simulation Tool, viz obr. 5.8) byl vyvinut ve výzkumném středisku firmy Rockwell Automation v Praze s původní intencí demonstrovat jednoduchou vizuální formou schopnosti a výhody použití agentové technologie v průmyslové automatizaci (Vrba, 2006). Systém implementovaný v jazyce Java a multiagentní platformě JADE je zaměřen na simulaci úlohy dopravy výrobků a materiálu ve výrobním závodě pomocí dopravních pásů nebo automatických samořízených vozíků (AGV). Každá dopravní komponenta, jako je výhybka, dopravní pás nebo výrobní jednotka je reprezentována pomocí agenta, který samostatně rozhoduje o lokálním řízení jemu svěřené výrobní komponenty a současně komunikuje s ostatními agenty prostřednictvím zasílání zpráv. Potřeba komunikace vyvstává v okamžiku, kdy agent není schopen svými vlastními prostředky vyřešit danou situaci, např. zareagovat na poruchu, nebo i v běžném provozu, kdy si agenti vyměňují informace o dopravovaných výrobcích, hledají optimální trasu pro přepravu, nebo rekonfigurují řídicí systém při přidání nebo odebrání komponent systému za běhu (Mařík a kol., 2003).



Obr. 5.8 Multiagentní simulační systém MAST.

Významným krokem směrem k možnostem reálného nasazení bylo doplnění systému MAST o rozhraní pro přístup k datové tabulce programovatelného automatu (viz odst. 5.2.1.1). To následně umožnilo experimentální řízení fyzické výrobní linky v laboratoři ACIN na Technické univerzitě ve Vídni. Zde je instalován zmenšený model dopravního systému (viz obr. 5.9) skládající se ze čtrnácti dopravních pásů, dvanácti výhybek, čtyř dokovacích stanic a jednoho PLC typu ControlLogix s vlastním řídicím systémem implementovaným v klasické žebříčkové logice. Tento řídicí systém byl nahrazen systémem agentním, provozovaným na PC, který s PLC komunikuje pomocí sítě Ethernet a přímo tak získává údaje ze senzorů a posílá signály k akčním členům. Experimentálně bylo ověřeno, že jsou agenti schopni přepravovat výrobky po optimálních trasách a efektivně reagovat na poruchy v dopravním systému vyhledáváním alternativních tras (Vrba a kol., 2008).



Obr. 5.9 Model výrobní linky v laboratoři ACIN na Technické univerzitě ve Vídni a jeho fyzické řízení agentním systémem MAST.

Z pohledu zpracování a výměny znalostí je původní architektura systému MAST vlastně rigidní a implicitní, bez známek využití jakékoliv sémantiky. Informace o okolním světě jsou v agentovi uloženy v lokálních proměnných a obsah zasílaných zpráv je ve formátu XML. Důvodem je především fakt, že kolem roku 2000, kdy byla implemen-

tace systému MAST zahájena, nebyly sémantické technologie a příslušné softwarové nástroje tak pokročilé, jako je tomu dnes. Nejnovějšími technologiemi té doby byly standardy organizace FIPA (Foundation for Intelligent Physical Agents), kompatibilní agentní platformy, jako JADE nebo FipaOS, a formát XML pro zápis strukturované informace. Hlavní nevýhodou bylo, že interpretace významu přijímaných zpráv byla pevně naprogramována v těle agenta. To komplikovalo možné rozšíření nebo adaptaci systému na nové podmínky, jež vyžadovaly přeprogramování stávajících agentů.

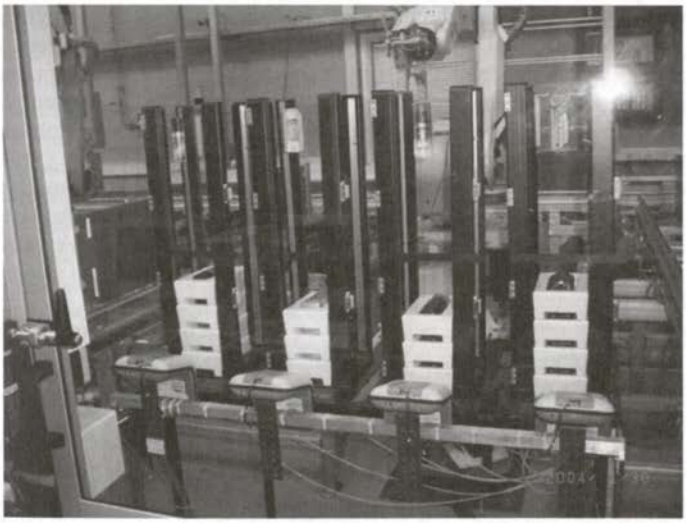
K překonání těchto problémů a především pro umožnění automatického zpracování znalostí a uvažování nad nimi byl systém MAST rozšířen tak, aby agenti používali explicitní sémantiku a ontologie pro uchovávání, zpracování a výměnu informací o uživatelských objednávkách, plánování výroby a dopravě materiálu. Využitím nově navržené ontologie a návodu pro její integraci v distribuovaném řídicím systému (viz odst. 5.4.2) byl naprogramován obecný agent objednávky, agent produktu, agent výrobní stanice, agent dodávky a agent dopravy spolu s interakčními scénáři umožňujícími výrobu libovolného, předem neznámého typu výrobku. Ukažme reálnou aplikaci, na níž byla funkčnost navrženého postupu ověřena.

5.5.2 Případová studie – balicí linka na Univerzitě v Cambridgi

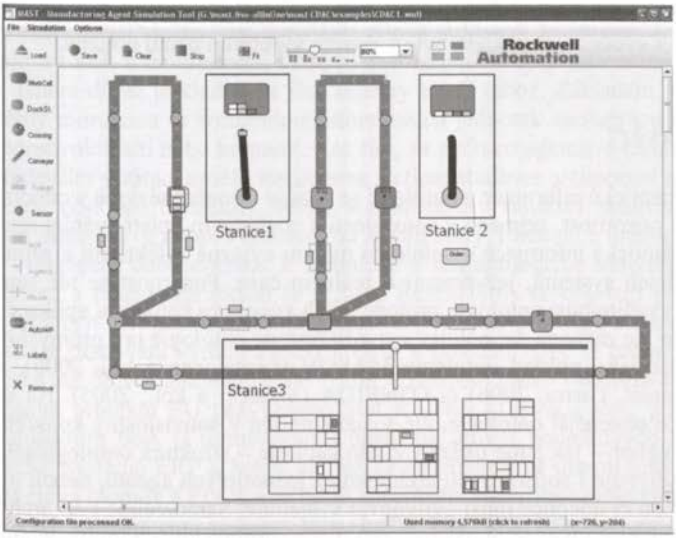
Laboratorní model balicí linky v laboratoři DIAL (Distributed Information and Automation Laboratory) na univerzitě v Cambridgi (Fletcher a kol., 2003) byl v předchozích letech použit jako reálný výrobní systém, na němž byla ověřována základní funkcionalita agentů řídicího systému MAST. Prototypová linka provádí balení dvou různých typů dárkových kosmetických balíčků, do nichž lze vložit libovolnou kombinaci tří kosmetických přípravků ze čtyř dostupných typů (holicího gelu, žiletky, pěny a deodorantu). Přípravky jsou uloženy ve vertikální čtyřslotové skladovací jednotce, odkud jsou odebírány robotem Fanuc M6i a vkládány do boxů (viz obr. 5.10). Linka obsahuje dvě takovéto nezávislé balicí stanice (označené Stanice1 a Stanice2, obr. 5.11). Prázdné boxy stejně tak jako hotové výrobky jsou ukládány ve skladovací stanici (Stanice3, obr. 5.11), která obsahuje portálového robota a tři policové skladovací jednotky. Doprava boxů mezi stanicemi je zajištěna jednokolejným dopravním systémem Montrac uspořádaným do jedné horizontální a dvou vertikálních smyček.

V původní realizaci simulace balicí linky laboratoře DIAL v systému MAST, bez použití sémantiky, byla základní množina agentů pro řízení dopravního systému rozšířena o nové typy agentů reprezentující jednotlivá zařízení balicí linky, jimiž byl agent robota Fanuc M6i, agent vertikální skladovací jednotky, agent portálového robota a agent policového skladu (Fletcher, Vrba, 2006). Navíc byl implementován agent objednávky pro příjem a vyřízení objednávky na dárkový balíček a agent produktu, jenž reprezentoval vlastní fyzický výrobek. Na rozdíl od nového způsobu nebyla zařízení seskupena do výrobních stanic, tj. agent produktu vyjednával o elementárních operacích s každým agentem zařízení zvlášť. Mezi těmito operacemi je vyjmutí prázdného boxu z policového skladu portálovým robotem, rezervace kosmetického přípravku ve vertikální skladovací jednotce, jeho přemístění robotem do boxu a uložení finálního produktu zpět na volnou pozici v policovém skladu. Pro vyjednávání mezi agentem produktu a ostatními agenty je použit protokol contract-net (Smith, 1980), přičemž seznam dostupných agentů poskytujících danou operaci je získáván vždy dynamicky, a to dotazem na agenta zlatých stránek. Tím je zajištěna flexibilita výrobního systému. Simulací bylo

ověřeno, že je balicí linku možné snadno rozšířit o další balicí stanici (jednotku s robotem a vertikálním skladem) společně s obslužnou smyčkou dopravního systému. Takto nově přidaní agenti pouze zaregistrují své služby u agenta zlatých stránek, a tím jsou plně integrováni do stávajícího systému. Chování produktového agenta, tj. vlastní algoritmus řízení výroby, není třeba jakkoliv modifikovat.



Obr. 5.10 Balicí linka v laboratoři DIAL na Univerzitě v Cambridgi.



Obr. 5.11 Simulace balicí linky DIAL v multiagentním systému MAST.

V nové sémantické realizaci řízení balicí linky bylo vytvořeno rozšíření obecné ontologie popsané v odst. 5.4.1 jako samostatný, pro výrobní linku specifický, modul. Toto rozšíření zahrnuje jednak třídy pro popis boxu a vkládaných kosmetických přípravků a jednak výrobní plán pro balení boxu. Plán sestává ze tří kroků, každého pro jeden slot v boxu, asociovaných s operací vkládání do boxu a vyžadujících vstupní materiál (prázdný nebo již částečně naplněný box a vlastní kosmetický přípravek).

Objednávky jsou do systému vkládány prostřednictvím speciální objednávkové komponenty v grafickém uživatelském rozhraní systému MAST. Jejím úkolem je zkomponovat sémantický popis objednávky ve formátu kompatibilním s navrženou ontologií. Na základě tohoto popisu je agentem, který spravuje výrobní plány, vytvořen plán ušitý na míru dané konkrétní objednávce. To se udělá tak, že se parametry z objednávky (typ vkládaného kosmetického přípravku) přenesou k odpovídajícím krokům výroby, popř. se některý krok z plánu vypustí, je-li daný slot požadován jako prázdný. Takovýto specifický plán se pak předá agentovi výrobku, který jej začne automaticky vykonávat. Pro každý krok kontaktuje agenty výrobních stanic, poskytujících operaci vkládání do boxu, s dotazem na nejbližší možný termín provedení operace. Agent výrobní stanice kontaktuje svého podřízeného agenta vertikální skladovací jednotky s dotazem, zda je k dispozici příslušný kosmetický přípravek, a pak dojedná s agentem dodávky dopravu boxu, ať již prázdného z policového skladovacího systému, nebo částečně naplněného z předchozí balicí stanice. Produktový agent pak vybere vítěznou balicí stanici a požádá ji o provedení nabídnuté operace.

Hlavní výhodou této realizace je možnost rozšířit výrobní systém o nové typy výrobků, popř. nové výrobní postupy, bez nutnosti měnit stávající řídicí systém, tj. chování agentů. Pro ověření byl v ontologii vytvořen plán pro „rozbalení" výrobku, v němž jde o výběr plného boxu ze skladu a vrácení jeho kosmetických přípravků zpět do vertikálních skladů v balicích jednotkách. Takovýto plán byl pak předložen produktovému agentu, který jej byl schopen vykonat bez jakékoliv úpravy jeho chování. Předpokladem bylo pouze to, že balicí jednotky měly naimplementovánu a ve zlatých stránkách zaregistrovánu operaci výběr z boxu, která byla v novém výrobním plánu použita.

5.6 Závěr

Využití sémantické informace a ontologií se věnuje v současné době v odborné komunitě mimořádná pozornost, zejména v souvislosti s postupným uplatňováním multiagentních řešení. Sémantická informace a ontologie mohou výrazně zefektivnit a zejména zrychlit činnost řídicích systémů, jež pracují v reálném čase. Pozornost se též soustřeďuje na opětovnou využitelnost ontologií, protože jejich vývoj pro konkrétní aplikace je relativně drahý. Proto lze zaznamenat pokusy vytvořit obecné ontologie pro průmyslovou výrobu. Uveďme zde jako příklad ontologie MASON (Lemaignan a kol. 2006), Automation Objects (Lopez, Lastra, 2006) či O3NEIDA (Vyatkin a kol., 2005). Na druhé straně vznikají sice obecnější ontologie, ale využitelné jen v souvislosti s konkrétním řídicím systémem, neboť – jak jsme ukázali v této kapitole – struktura ontologického subsystému silně ovlivňuje i softwarovou architekturu jednotlivých agentů, neboli obě architektury se mohou či dokonce musí ovlivňovat vzájemně. Samozřejmě tyto architektury pak následně ovlivňují i budování nástrojů pro tvorbu průmyslových multiagentních systémů (Tichý a kol., 2012).

Pro některé konkrétní aplikace tak vznikly nové agentové architektury, které využívají ontologické struktury jako hlavní formát pro reprezentaci a uchovávání znalostí, pro jejich výměnu mezi agenty i pro automatizované vyvozování znalostí nebo informací nových. V sémantickém rozšíření systému MAST (Vrba a kol., 2011) se řešení soustřeďuje na dynamickou alokaci zdrojů řízenou ontologií, která popisuje jak objednávky, tak i výrobní procesy (sled operací, požadovaný materiál nebo polotovary apod.). Jinou multiagentní architekturu řídicího systému, která je ovlivněna specifickou ontologií pro přepravu materiálu, lze najít v článku (Vallée a kol. 2011). Agenti využívají tzv. *situační model*, zachycující reálný stav výrobního prostředí, a tzv. *model aktivit*, který poskytuje na základě změn v situačním modelu agentovi informaci o tom, jaká pozorování by mohl v nejbližších okamžicích očekávat a na ně se připravit.

Dalším krokem v integraci architektur ontologií a řídicích systémů je trend vedoucí k vytváření mnohdy složitých a informačně bohatých *vnitřních modelů světa*, které odrážejí změny v okolí tak, že do značné míry agentovi umožňují porozumět celkové současné situaci v dostatečné složitosti jako podklad pro dynamický výběr nejvhodnější strategie nebo modelu chování. Takové vnitřní modely světa jsou prezentovány například v práci (Lepuschitz a kol., 2011). Ty obsahují nejen situační model a model aktivit, ale též databázi pozorování a očekávání. Dalším rozšířením jsou ontologie, které obsahují nejen deklarativní fakta, ale též explicitně specifikované modely chování agenta v procedurálním tvaru (Radakovič a kol., 2012). Takovýto typ ontologií umožňuje obohatit vnitřní model světa o seznam možných modelů chování a zejména dynamické modifikování tohoto seznamu. Tvorba takovéhoto modelu je však relativně velmi pracná.

V souvislosti se sémantickou informací a multiagentními řešeními se často diskutuje o paradigmatu softwarových řešení založených na architekturách SoA (*Service-Oriented Architectures*). Jde o technologie pro návrh znovu použitelných, volně vázaných a vzájemně interoperabilních modulů. Architektury SoA se zaměřují především na definování rozhraní, protokolů a formát dat pro zpřístupňování služeb s cílem umožnit vývojářům kombinovat nejrůznější služby pro vytvoření cílové aplikace. Je zřejmé, že filosoficky jsou architektury SoA a multiagentní systémy založeny na stejných či velmi podobných principech. Jenom důraz je kladen na jiné aspekty téměř téhož. Základem je totiž vždy soubor tvořený množinou či komunitou autonomních jednotek spolupracujících v prostředí kolaborativních sítí nebo komunit. Lze říci, že zatímco agentové technologie byly vyvíjeny především v rámci umělé inteligence s cílem studovat autonomní chování jednotlivých entit, obdobné myšlenky, řešení a architektury vznikaly nezávisle v oblasti softwarového inženýrství, avšak s důrazem na architektury, rozhraní a interoperabilitu.

Teprve v poslední době dochází k technické a terminologické konvergenci multiagentních řešení a řešení opírajících se o SoA. Řešení SoA se objevují především na nejvyšších úrovních tzv. „business intelligence" a postupně pronikají směrem k nižším úrovním, jako je plánování výroby a rozvrhování. V poslední době se začínají objevovat i přístupy SoA na úrovni automatického řízení v reálném čase. Takovéto úsilí se stalo například náplní projektu EU nazývaného SOCRADES (Taisch a kol., 2011), kde se využívá architektura SoA (i terminologie) nejen na úrovni podnikových plánovacích a rozvrhovacích systémů ERP a MES, ale též až na přístrojové úrovni (chytré senzory, embedované řídicí moduly a PLC), a to tímto způsobem: specifické funkcionality jednotlivých fyzických zařízení jsou chápány jako služby, které jsou nabízeny dalším službám v dané továrně. Můžeme se tedy na toto úsilí dívat jako na evoluci holonické agentové

architektury, kde řídicí aktivity na úrovni fyzických zařízení jsou chápány jako služby poskytované jiným agentům. Jiný přístup (Cândido a kol., 2011) se snaží o kombinaci agentové technologie a služeb SOA a pracuje s konceptem *Service-Oriented Agents*. Takovéto koncepty však představují spíše terminologický a konceptuální kompilát než skutečně technicky inovativní řešení.

Kromě aplikace multiagentních systémů a sémantické či ontologické informace pro potřeby automatického řízení v reálném čase se v poslední době prudce zvyšuje zájem průmyslových institucí o plánovací, rozvrhovací a optimalizační systémy na úrovni podniků a dílen či jiných složitých dopravních a zásobovacích systémů. Hlavním důvodem je vysoká ekonomická efektivita potenciálně úspěšných řešení. Velmi dobrý přehled stávajícího stavu je uveden v práci (Skobelev, 2011). Kromě plánování a rozvrhování pro běžné, relativně „stabilní" výrobní procesy je zájem zejména o multiagentní řešení složitých optimalizačních úloh v těch výrobních úsecích, kde jde o vysokou flexibilitu a složitost prostředí a o vysokou potřebu dynamických změn. Typickým problémem je například náběh výroby letadel, automobilů či turbín či zásobování kosmických stanic. Jinou důležitou úlohou je řízení tzv. *chytrých energetických sítí* (smart grids). Zde všude lze očekávat přímý či téměř přímý transfer multiagentových technologií spojených se zpracováním sémantické informace do navrhovaných řešení, viz např. (Strasser a kol., 2011).

Výzkum možností využití sémantické informace a ontologií v oblasti průmyslové automatizace a průmyslové výroby významně přispěl k aplikovatelnosti myšlenek multiagentních systémů v průmyslu. Má však mnohem širší dopad než jenom v průmyslové výrobě. Přináší totiž řešení bezprostředně a jednoduše aplikovatelná i v dalších úlohách a oblastech. Přináší tedy posun technologií v širším měřítku.

Literatura

ANSI/ISA-88.01.1995: *Batch Control Part 1: Models and Terminology*. American National Standard, The Instrumentation, Systems and Automation Society, 1995.

Al-Safi Y., Vyatkin V.: An Ontology-Based Reconfiguration Agent for Intelligent Mechatronic Systems. In: *Holonic and Multi-Agent Systems for Manufacturing, LNAI 4659*. Springer-Verlag, Berlín, Heidelberg 2007, s. 114–126.

Andreev M., Rzevski G., Skobelev P., Shveykin P., Tsarev A., Tugashev A.: Adaptive planning for supply chain networks. In: *HoloMAS 2007, LNAI 4659*. Springer-Verlag Berlín Heidelberg, 2007, s. 215–224.

Brennan R., Vrba P., Tichý P., Zoitl A., Sünder C., Strasser T., Mařík V.: Developments in Dynamic and Intelligent Reconfiguration of Industrial Automation, *Computers in Industry*, vol. 59/6, 2008, s. 533–547.

Bussman S., Schild K.: An Agent-based Approach to the Control of Flexible Production Systems. In: *Proceeding of 8th IEEE International Conference on Emerging Technologies and Factory Automation*. Antibes Juan-les-pins, 2001, s. 169–174.

Camarinha-Matos L. M.: Multi-Agent Systems. In: *Proceedings of the International Conference on AI, Simulation and Planning in High Autonomy Systems*. SCS Publication, Lisabon 2002, s. 27–36.

Cândido G., Barata J. A.: Multiagent Control System for Shop Floor Assembly. In: *Holonic and Multi-Agent Systems for Manufacturing, LNAI 4659*. Springer-Verlag, Berlín, Heidelberg, 2007, s. 293–302.

Cândido G., Colombo A. W., Barata J., Jammes F.: Service-Oriented Infrastructure to Support the Deployment of Evolvable Production Systems, *IEEE Trans. Industrial Informatics*, 7(4), 2011, s. 759–768.

Dean M., Schreiber G.: *OWL Web Ontology Language Reference [online].* World Wide Web Consortium Recommendations, 2004, http://www.w3.org/TR/owl-ref/.

Fletcher M., Brusey J.: The Story of the Holonic Packing Cell. In: *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems*. ACM Press, Melbourne 2003, s. 342–348.

Fletcher M., Vrba P.: A Brace of Agent Simulation Scenarios. In: *Proceedings of the IEEE Workshop on Distributed Intelligent Systems*, Praha 2006, s. 169–176.

Foundation for Intelligent Physical Agents, 2002. http://www.fipa.org.

Grüninger M., Kopena J. B.: Planning and the Process Specification Language. In: *Proceedings of WS2 ICAPS*, 2005, s. 22–29.

Gruber T.: A Translation Approach to Portable Ontologies. *Knowledge Acquisition*, 5(2), 1993, s. 199–220.

Hellingrath B., Witthaut M., Böhle C., Brügger S.: An Organizational Knowledge for Automotive Supply Chains. In: *Holonic and Multi-Agent Systems for Manufacturing, LNAI 5696*. Springer-Verlag, Berlín, Heidelberg, 2009, s. 37–46.

Kadera P., Tichý P.: Plan, Commit, Execute Protocol in Multi-agent Systems. In: *Holonic and Multi-Agent Systems for Manufacturing, LNAI 5696*, Springer-Verlag, Berlín, Heidelberg, 2009, s. 155–164.

Lepuschitz W., Vallée M., Merdan M., Vrba P., Resch J.: Integration of a Heterogeneous Low Level Control in a Multi-agent System for the Manufacturing Domain. In: *Proc. IEEE Conf. on Emerging Technologies and Factory Automation.* Mallorca, 2009, s. 7–14.

Lepuschitz W., Zoitl A., Vallée M., Merdan M.: Towards Self-Reconfiguration of Manufacturing Systems using Automation Agents. *Transactions on Systems, Man, and Cybernetics*, Part C: Applications and Reviews, 41(1), 2011, s. 52–69.

Lemaignan S., Siadat A., Dantan J. Y., Semenenko, A.: MASON: A Proposal for an Ontology of Manufacturing Domain. In: *Proc. of the IEEE Workshop on Distributed Intelligent Systems*. IEEE Computer Society Press, 2006, s. 195–200.

Lopez O., Martinez Lastra J. L.: Using Semantic Web Technologies to Describe Automation Objects, *Int. J. Manufacturing Research*, vol. 1, 4, 2006, s. 482–503.

Mařík V., McFarlane D. C.: Industrial Adoption of Agent-based Technologies. *IEEE Intelligent Systems*, 20(1), 2005, s. 27–35.

Mařík V., Vrba P., Pěchouček M.: *Od holonů k virtuálním organizacím*. Umělá inteligence 4, Academia, Praha, 2003, s. 407–447.

Manola F., Miller E.: *RDF Primer (online).* World Wide Web Consortium Recommendations, 2004, http://www.w3.org/TR/rdf-primer/.

McLean C., Lee Y. T., Shao G., Riddick F.: Shop Data Model and Interface Specification, *NISTIR 7198*, 2005.

Merdan M., Koppensteiner G., Hegny I., Favre-Bulle B.: Application of an Ontology in a Transport Domain. In: *Proc. Of the IEEE International Conference on Industrial Technology*. Sichuan University, Chengdu, Čína, 2008, s. 1–6.

Meyer G., Främling K., Holmström J.: Intelligent products: A Survey. *Computers In Industry*, vol. 30, 3, 2009, s. 137–148.

Obitko M., Vrba P., Mařík V., Radakovič M.: Semantics in Industrial Distributed Systems. In: *Proceedings of the 17th IFAC World Congress*. Soul 2008, s. 13880–13887.

Pěchouček M., Rehák M., Charvát P., Vlček T., Kolář M.: Agent-Based Approach to Mass-Oriented Production Planning: Case Study. *IEEE Transactions on Systems, Man, and Cybernetics*, Part C: Applications and Reviews, nol. 37, 3, May 2007, s. 386–395.

Pěchouček M., Vokřínek J., Bečvář P.: ExPlanTech: Multiagent Support for Manufacturing Decision Making. *IEEE Intelligent Systems*, nol. 20, 1, 2005, s. 67–74.

Radakovič M., Obitko M., Mařík V.: Dynamic Explicitely Specified Behaviors in Distributed Agent-Based Industrial Solutions. *Journal of Intelligent Manufacturing*, vol. 23, 6, 2012, s. 2601–2621.

Runde S., Fay A.: Software Support for Building Automation Requirements Engineering – An Application of Semantic Web Technologies in Automation. *IEEE Trans. Industrial Informatics*, 7(4), 2011, s. 723–733.

Rzevski G., Skobelev P., Andreev V.: MagentaToolkit: A Set of Multi-agent Tools for Developing Adaptive Real-Time Applications. In: *Holonic and Multi-Agent Systems for Manufacturing, LNAI 4659*, Springer-Verlag, Berlín, Heidelberg, 2007, s. 303–313.

Skobelev P.: Multi-Agent Systems for Real Time Resource Allocation, Scheduling, Optimization and Controlling: Idustrial Appliactions. In: *Holonic and Multi-Agent Systems for Manufacturing, LNAI 6867*, Springer-Verlag, Berlín, Heidelberg, 2011, s. 1–14.

Smith R. G.: The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, C-29(12), 1980, s. 1104–1113.

Strasser T., Andren F., Stifter A.: A Test and Validation Approach for the Standard-Based Implementation of Intelligent Electronic Devices in Smart Grids. In: *Holonic and Multi-Systems for Manufacturing, LNAI 6867*, Springer-Verlag, Berlín, Heidelberg, 2011, s. 50–61.

Taisch M., Colombo A. W., Karnouskos S., Cannata A.: *SOCRADES Roadmap: The Future of SOA-based Factory Automation,* 2010, http://www.socrades.eu/.

Tichý P., Kadera P., Staron R. J., Vrba P., Mařík V.: Multi-agent System Design and Integration via Agent Development Environment. *Engineering Applications of Artificial Intelligence*, vol. 25, 4, 2012, s. 846–852.

Vallée M., Merdan M., Lepuschitz W., Koppensteiner G.: Decentralized Reconfiguration of a Flexible Transportation System. *IEEE Transactions on Industrial Informatics*, 7(3), 2011, s. 505–516.

Van Leeuwen E. H., Norrie D.: *Intelligent Manufacturing: Holons and Holarchies. Manufacturing Engineer*, vol. 76, 2, 1997, s. 86–88.

Vrba P.: *Simulation in Agent-based Control Systems: MAST Case Study, International Journal of Manufacturing Technology and Management*, vol. 8, 1/2/3, 2006, s. 175–187.

Vrba P., Mařík V.: *Capabilities of Dynamic Reconfiguration of Multiagent-Based Industrial Control Systems.* IEEE Transactions on Systems, Man, and Cybernetics - part A: Systems and Humans, Vol. 40, No. 2, 2011, s. 213–223.

Vrba P., Mařík V., Merdan M.: Physical Deployment of Agent-based Industrial Control Solutions: MAST Story. In: *Proceedings of the IEEE International Conference on Distributed Human-Machine Systems*, Athény 2008, s. 133–139.

Vrba P., Radakovič M., Obitko M., Mařík V.: Semantic Technologies: Latest Advances in Agent-based Manufacturing Control Systems, *International Journal of Production Research*, 49(5), 2011, s. 1483–1496.

Vyatkin V.: *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design*. ISA, 2007.

Vyatkin V., Christensen J., Martinez Lastra J. L., Auinger F.:OOONEIDA: An Open, Object-Oriented knowledge economy for intelligent industrial automation, *IEEE Trans. Industrial Informatics*, vol. 1, 1, 2005, s. 4–17.

6 Automatické sémantické anotování a extrakce informací

6.1 Využití extrakce informací a sémantického anotování

6.1.1 Úvod

Metody umělé inteligence v širokém smyslu, včetně například odvozování nad ontologickými modely, kterými se vesměs zabývaly předchozí kapitoly, jsou z velké části založeny na využívání strukturovaných znalostí a dat. Jak jsme však lehce nastínili již v kapitole 2 věnované sémantickému webu, v mnoha případech se strukturované znalosti a data prolínají se znalostmi a daty vyjádřenými v textové podobě – ať již proto, že informace primárně vznikají v podobě textových dokumentů, nebo proto, že původně strukturovaná data byla zformátována do textové podoby kvůli čitelnosti pro lidského uživatele a k jejich výchozí podobě (již) není přístup. Tím se otvírá široký prostor pro metody *extrakce informací* (information extraction, IE), které automaticky vytvářejí strukturovaná data z textových, popř. pro *sémantické anotování* textů, jež přiřazuje částem textu sémantiku. Je dobré si uvědomit, že v konečném důsledku jde v podstatě o totéž (s výjimkou toho, že sémantické anotování může být prováděno i ručně, takže jde o pojem obecnější); při extrakci informací je totiž zachování vazby vyextrahovaných dat a jejich sémantiky na původní text, a tudíž realizace sémantické anotace tohoto textu, jen technickou otázkou.

Různé typy úloh v rámci extrakce informací a sémantického anotování, ale i metody jejich řešení, jsou náplní této kapitoly. V úvodní části kapitoly (odstavce 6.1.2 a 6.1.3) se pokusíme stručně charakterizovat *pojem* „sémantické anotování" na obecné úrovni a popsat různé *typologické* dimenze, podle nichž je možné jeho různé varianty zařadit. Následuje sekce věnovaná sémantické anotaci prováděné *ručně* člověkem, ač často v určité vazbě na anotování automatické, oddíl 6.2. Zbytek kapitoly se již zaměřuje na *automatickou* extrakci informací. Oddíl 6.3 rozebírá typy *úloh* extrakce informací, a také způsoby, jimiž lze *hodnotit* jejich úspěšnost. Oddíl 6.4 přináší systematický přehled tří hlavních kategorií extrakčních metod, buď založených na *ručně vytvářených* pravidlech (popř. komplexnějších modelech, někdy označovaných i jako „extrakční ontologie"), nebo na *induktivním učení* a na analýze *pravidelného formátování* (tzv. „wrapperový" přístup); vedle toho jsou uvedeny i přístupy usilující o snížení náročnosti pořizování anotovaných trénovacích příkladů: *aktivní učení* a *bootstrapping*. Oddíl 6.5 detailněji rozebírá a na příkladech ilustruje *vybrané výpočtové metody*, konkrétně využívání extrakčních ontologií, transformačních pravidel a grafických modelů. V oddílu 6.6 najde čtenář specifické zamyšlení nad perspektivami využívání extrakce informací pro hro-

madný převod textových informací obsažených na webu do formátu sémantického webu. V závěrečném oddílu 6.7 pak čtenář kromě shrnující poznámky nalezne také odkazy na významné *tuzemské projekty* spadající do dané problémové oblasti.

6.1.2 Anotování a sémantika

Samotný pojem „anotovat" lze vykládat (podle latinského „ad-notare") jako „vytvořit poznámku k (něčemu)". Jde tedy vesměs o vytvoření nějakého sekundárního informačního zdroje, který se vztahuje ke zdroji primárnímu. Anotování se používá v širokém spektru oborů a také mnoha rozličnými způsoby. V tomto textu nemáme prostor na výčet (ani nejrozšířenějších) konkrétních anotačních metod a nástrojů. Zaměříme se spíše na obecné principy a typologii různých pojetí anotování zdrojů, zejména s ohledem na anotaci *sémantickou* jako hlavní předmět kapitoly.

Význam adjektiva „sémantický", vůči kterému se určitým způsobem vymezuje velká část tohoto svazku, je ještě komplikovanější; viz rozbor v kapitole 1. Pro účely této kapitoly se omezíme na dva jeho ustálené významy:

Za sémantická data jsou v informatice obecně považována taková, která explicitně odkazují na svůj „význam", tj. k jejich pochopení nejsou potřebné nějaké dodatečné znalosti, které jejich příjemce mít může či nemusí. Aby například elementární údaj „160/95" mohl být chápán jako sémantický, musí být přímo v jeho reprezentaci zajištěno, že spolu s ním bude vždy dostupná i informace o tom, že jde o hodnotu krevního tlaku pacienta naměřeného lékařem v nemocnici. Sémantika elementárních dat je tedy zajištěna pomocí *odkazu* na nějaký opakovaně využívaný, známý pojem (či více pojmů).

Z teoretičtějšího pohledu je sémantikou určitého výrazu uloženého v informačním systému jeho formální a výpočetní interpretace. Pojem sémantiky zde stojí v určitém kontrastu s pojmem syntaxe. Například sémantika výrazu „x AND x AND y" bude typicky totožná se sémantikou syntakticky odlišného výrazu „x AND y" a bude jí (v určitém zjednodušení) množina objektů, které splňují jak podmínku x, tak podmínku y. Obsah množiny se vypočítá jako průnik dvou množin odpovídajících výrazům x a y. Sémantiku není možné spolehlivě zjistit bez zapojení *odvozovacího mechanismu*.

Z hlediska prvního významu můžeme jako sémantickou anotaci chápat takový sekundární dokument, který je spojen nejen s primárním dokumentem, ale také s dalším zdrojem (nebo více zdroji), který vymezuje význam jeho prvků – takový zdroj budeme označovat jako *anotační model*. Z hlediska druhého významu je pak sémantickou anotací soubor tvrzení ve formálním jazyce, kterým lze pomocí odvozovacích postupů přiřadit interpretaci. Mapování mezi množinou možných syntaktických podob tvrzení a množinou možných interpretací tedy není typu „1 : 1" ale „N : M"; tj. stejný význam lze vyjádřit více syntaktickými způsoby a současně jedno syntaktické vyjádření může být mnohoznačné. Vzorovým příkladem sémantické anotace „v obou významech" jsou proto sekundární dokumenty sémantického webu (popisující například webové sídlo nebo multimediální prezentaci jako dokument primární), které jsou vyjádřeny ve formálním jazyce (jakým je RDF[1]) a jejichž prvky se odvolávají na pojmy ze strukturovaného anotačního modelu s charakterem ontologie (např. v jazycích RDFS nebo OWL); o těchto jazycích bylo pojednáno podrobněji v kapitolách 2 a 4.

[1] RDF, Resource Description Framework, http://www.w3.org/RDF (19. 12. 2010), viz kap. 2.

Doplňme ještě, že ačkoliv anotování historicky vznikalo v éře papírových dokumentů, naplnění podstaty sémantického anotování si lze v případě elektronicky nečitelných poznámek představit jen stěží. V dalším textu budeme proto automaticky předpokládat, že anotace, anotační model a zpravidla i primární (anotovaný) dokument mají elektronickou povahu.

6.1.3 Stručná typologie anotačních přístupů

Ze samotné podstaty pojmu anotování plyne většina z kritérií, podle kterých můžeme anotační „situace" členit:
- Co je anotováno? (Charakter primárního informačního zdroje.)
- Kdo anotuje? (Charakter autora anotace a jeho vztah k primárnímu dokumentu.)
- Čím se anotuje? (Charakter anotačního modelu jako zdroje prvků anotace a způsob, jakým se z těchto prvků anotace sestaví.)
- O čem anotace vypovídá? (Vztah anotace k primárnímu dokumentu.)
- Kde je anotace uložena? (Umístění anotace nebo odkazu na ni.)

6.1.3.1 Co je anotováno

Jako první se zřejmě nabízí hledisko *modality informačního obsahu* primárních dokumentů. Tradičním objektem anotování jsou dokumenty *textové*. Lze však anotovat také dokumenty *obrazové* (obrázky, videa), *zvukové* nebo obecně *multimediální*.

Dále můžeme vzít do úvahy *granularitu* takových dokumentů ve vztahu k anotaci. Anotace se může vztahovat:

- K dokumentu *jako celku* – předpokládáme dokument jistým způsobem „globálně" identifikovaný, např. stránku HTML nebo dokument uchovávaný v systému pro správu dokumentů.
- K *ucelené části* dokumentu identifikované „lokálně" – může jít například o sekci textového dokumentu nebo o ohraničený záběr videa.
- K *fragmentu* dokumentu, jehož hranice jsou vymezeny teprve anotováním samotným – označený blok textu, výřez obrázku, ad hoc posloupnost rámců videa apod. Hranice anotace jsou v případě textu zpravidla určeny značkami, mluvíme proto často o *značkování* (mark-up) dokumentu.

Proces anotování ovšem někdy i v případě, že předmětem anotování jsou celé (textové) dokumenty, zahrnuje v první fázi výběr fragmentů, které pak slouží jako zdroj prvků anotace samotné. Triviálním příkladem je automatické anotování dokumentů pomocí výběru významových slov z jejich obsahu, které je prvním krokem automatického indexování rutinně používaného při tvorbě plnotextových databází. V jednoduchém případě se vztah mezi slovy jako prvky anotace a konkrétními místy jejich výskytu zcela ztrácí. Naopak jako skutečné značkování fragmentů s jemnou granularitou můžeme chápat tvorbu lingvistických korpusů (viz například projekt Český národní korpus[2]), kde anotace zachycují morfologické charakteristiky jednotlivých slov, následně pak syntaktickou skladbu vět atd.

[2] Český Národní Korpus, Ústav Českého národního korpusu FF UK, http://www.korpus.cz (19. 12. 2010).

6.1.3.2 *Kdo anotuje*

Anotace může primárně vytvářet buďto člověk s větší či menší mírou softwarové podpory, nebo automatický proces (s možností korekčního vstupu od lidského uživatele). V převážné části této kapitoly se zaměřujeme na anotování *automatické*, které využívá prostředky disciplín tradičně chápaných jako podobory umělé inteligence: zpracování přirozeného jazyka, strojové učení, popř. zpracování obrazu a další. Jak již jsme naznačili v úvodu, automatické sémantické anotování má (zejména v případě textového charakteru primárních dokumentů) úzkou souvislost s extrakcí informací.

Ruční anotování však má také stále své uplatnění, zejména jako prostředek pro pořizování *trénovacích dat* pro anotátory automatické nebo při zpracování volněji strukturovaných avšak informačně bohatých dokumentů (např. právních norem nebo lékařských doporučených postupů), ze kterých je třeba znalosti co nejpodrobněji „vytěžit" do formalizované znalostní báze.

V některých scénářích je autor anotace totožný s autorem primárního dokumentu. Pokud sémantická anotace vzniká souběžně s dokumentem, mluvíme o sémantické tvorbě dokumentů („semantic authoring"); jindy může autor anotaci doplnit dodatečně. Na druhé straně je ale možné, aby autor anotace byl odlišný od autora dokumentů, a dokonce s ním ani nebyl v přímém kontaktu – tento model je obvyklý v případě „knihovnického" pojetí anotování.

6.1.3.3 *Čím se anotuje*

Z historické perspektivy je nejobvyklejším typem anotace text v přirozeném jazyce. *Anotačním modelem* může být v takovém případě celá *slovní zásoba* daného jazyka. Opačným extrémem je anotační model obsahující pevně danou množinu hodnot. Takový model lze chápat jako soubor kategorií dokumentů – anotace se pak redukuje na *kategorizaci* (klasifikaci). Triviálním příkladem je knihovnické rozlišení periodických a neperiodických publikací jako „přirozených" tříd dokumentů. Kategorizace se však může vztahovat i k obsahu dokumentu (např. při zařazování dokumentů do témat v rámci webových portálů), popřípadě k významu fragmentů textu (např. při rozpoznávání pojmenovaných entit typu „osoba", „organizace" či „místo") nebo oblastí v rámci obrázku (např. třídy „auto", „obličej", „zbraň").

„Střední cestu" představuje anotační model obsahující určité předem zvolené prvky, ale do kterého je dále možné přidávat prvky nové. Takový model je z hlediska *sémantického* anotování nejzajímavější. Prvky modelu jsou pojmy z určitého oboru a vztahy mezi nimi – model má tedy síťový charakter, který mimo jiné usnadňuje vyhledávání. Tím se dostáváme k modelům typu *tezauru* nebo *ontologie*, které byly rozebrány zejména v kapitole 1. Připomeňme, že základním rozdílem mezi nimi je terminologický charakter tezauru ve srovnání s „věcným" charakterem ontologie. Prvky tezauru jsou v zásadě *termíny* a jejich propojení má primárně charakter vztahů „nahraditelnosti" mezi slovy, tj. synonymie či hyperonymie. Prvky ontologie jsou oproti tomu spíše *pojmy* (koncepty, třídy) zastupující množiny objektů (individuí, instancí) z reálného světa a vazby mezi ontologickými pojmy vyjadřují na jedné straně *množinové operace* nad množinami objektů odpovídající těmto pojmům (jde zejména o subsumpci a ekvivalenci), na straně druhé pak možné či nutné *vztahy mezi reálnými objekty* (např. v jazyce OWL toto zajišťují objektové vlastnosti jakožto binární relace). Typickým rysem ontologie je také ri-

gorózní *formalizace*, tj. vazba na logický kalkul s přesně popsanými vlastnostmi, umožňující odvozování.

Dosud jsme se zabývali strukturou anotačního modelu. Nyní se podívejme na strukturu *anotace samotné*, která je samozřejmě se strukturou anotačního modelu korelovaná, ale ne absolutně. Všimněme si, že použití slovní zásoby přirozeného jazyka může vést na anotaci zcela nestrukturovanou (prostý seznam klíčových slov), ale také na anotaci strukturovanou na úrovni větné skladby (v případě anotování volným textem); v principu si lze představit i aplikaci předdefinovaných vazeb na klíčová slova navržená ad hoc. Kategorizace do „přirozených tříd" sama pro anotaci žádnou strukturu nezakládá. V případě tezauru nebo ontologie je ovšem možné přenést do anotace přímo část struktury z modelu (u ontologie v jazyce OWL může jít například o výskyt přiřazení instance k některé třídě nebo o vyhlášení vztahu mezi dvěma instancemi určitých tříd): pak mluvíme o *relační anotaci* a ve specifickém případě automatické extrakce informací z textu jde o „relační extrakci" – viz tab. 6.1 v odst. 6.3.2). Na druhou stranu i vysoce strukturovaný anotační model lze využít jako pouhý „zásobník" izolovaných pojmů vkládaných do anotace.

Je zjevné, že „jednotková" anotace (vytvořená při jednom sezení jedním autorem pro jeden zdroj) může obsahovat prvky z různých anotačních modelů, a také mít různou míru vnitřní strukturovanosti. Pro příklad nemusíme chodit daleko, stačí se podívat na záznam v některé bibliografické databázi, který bude obsahovat jak předem dané „přirozené" kategorie, tak i volně tvořená klíčová slova a někdy i pojmy z tezauru.

6.1.3.4 O čem anotace vypovídá

Častým účelem anotování je charakterizovat *obsah* primárního dokumentu. Do této oblasti spadají jak klíčová slova, tak anotace ve volném textu a popis termíny neboli pojmy z doménového tezauru, popř. ontologie. Smysl však má také anotování informacemi o *kontextu* dokumentu, zejména o jeho vzniku a dalším životním cyklu – může jít o informace o autorovi, místě vzniku, předchozí verzi dokumentu apod. Třetím hlavním možným účelem anotování je *hodnocení* dokumentu. Anotace se může specificky věnovat úrovni kvality, resp. zajímavosti primárního dokumentu – charakteristickým případem je přiřazování „hvězdiček" webovým zdrojům. Z praktických důvodů je ovšem hodnocení často spojováno s charakterizací obsahu.

6.1.3.5 Kde je anotace uložena

V případě značkovacích přístupů bývá často anotace v plném rozsahu součástí *primárního dokumentu*. Příkladem je využívání standardu RDFa, popsaného v kapitole 2, odst. 2.2.6. Anotace může také být od dokumentu fyzicky oddělena a připojena k němu *odkazem* umístěným například v jeho hlavičce. Toto je časté u anotací typu komentáře či hodnocení.

Zásadně odlišnou možnost představuje umístění anotace (resp. odkazu na primární dokument) přímo do *anotačního modelu*. V jednoduché variantě tuto volbu představují webové katalogy typu Yahoo! nebo Seznam, které ze stránky určitého tématu (které můžeme s určitou licencí považovat za ontologický pojem) odkazují na stránky, jež se tohoto tématu týkají. Ve světě znalostních aplikací takový přístup reprezentuje model Topic Maps[3], v němž jsou jednotlivým tématům (vytvořeným například pro významné osob-

[3] Topic Maps jsou popsány standardem ISO 13250; http://www.topicmaps.org (27. 3. 2011).

nosti, místa, události apod.) v dané „mapě témat" přiřazeny tzv. externí výskyty – odkazy do „vrstvy dokumentů". Hodnotou odkazu do vrstvy dokumentů může být například webová stránka osoby, stránky s jejím životopisem nebo její fotografie, přičemž povaha cílového dokumentu se odráží již v typu odkazu, který na něj vede.

Poslední možností je udržovat anotace *odděleně* od primárních dokumentů i anotačního modelu. Tato možnost zbývá jako jediná v případě, že autor anotace nemá možnost editovat ani primární dokument, ani anotační model.

6.2 Scénáře pro ruční anotování

I když je naše pozornost zaměřena především na automatické extrakční metody, je několik důvodů, proč je třeba anotovat také ručně. Tři z nich jsou podrobněji rozebrány v následujícím výkladu. Je však nutné podotknout, že čistě ruční nástroje najdeme již jen zřídka; většina anotačních prostředí totiž bývá vybavena alespoň jednoduchou formou učícího se algoritmu. Nejjednodušším příkladem takového algoritmu je automatické předanotování každé fráze podle jejího nejčastějšího anotovaného významu v dosud anotovaných dokumentech.

6.2.1 Sémantická tvorba dokumentů

V tomto případě jsou anotace vytvářeny spolu s dokumentem a obvykle stejným autorem. Někteří autoři webových stránek doplňují své stránky sémantickými anotacemi popisujícími obsah stránky. V současnosti nejrozšířenějším anotačním modelem, jehož prvky lidé přidávají především do svých vlastních domovských stránek, je FOAF (Friend Of A Friend)[4] umožňující vzájemně odkazovat na své přátele, zájmy, navštívená místa apod. Technicky bývají anotace FOAF, ve formátu RDF, uloženy v separátním souboru „`foaf.rdf`" vedle vlastních stránek HTML, které na tato metadata odkazují pomocí metaznačky HTML `<link/>`. Alternativně mohou být profily FOAF uloženy u externích poskytovatelů. V každém případě jsou tyto profily pravidelně prohledávány a indexovány různými vyhledávači, které ve výsledné sociální síti dokáží vyhledávat a navigovat. Výhodou oproti plnotextovému vyhledávání je možnost strojově interpretovat údaje z profilů podle jejich významu. Vyhledávací server pro profily FOAF, www.foaf-search.net, uvádí k 19. 12. 2010 zhruba 4 miliony pokrytých osob a organizací. Internet však k tomuto datu má podle www.worldwidewebsize.com nejméně 13,6 miliard stránek; doména pokrytá jazykem FOAF je navíc poměrně úzká. Profil FOAF lze vytvořit celkem jednoduše vyplněním online formuláře[5] a stažením výsledných dat RDF. Všimněme si že, technicky vzato, profil FOAF nemusí nutně korespondovat s textovým obsahem příslušných webových stránek.

Anotační systém Ontomat (Handschuh, Staab, 2002) je nástrojem sloužícím k anotaci konkrétní webové stránky instancemi z ontologie, která modeluje libovolnou doménu. Uživatelské rozhraní sestává z jednoduchého editoru ontologií a webového prohlížeče. Uživatel může editovat koncepty v ontologii, přidávat do ní instance a propojovat tyto instance se zobrazenou webovou stránkou. Výsledkem je opět fragment RDF obsahující

[4] Specifikace jazyka FOAF: http://xmlns.com/foaf/spec (19. 12. 2010), viz též kap. 2, odst. 2.3.1.
[5] http://www.foaf-o-matic.org

výslednou anotaci, kterou lze vložit přímo do webové stránky nebo do propojeného souboru RDF. Původně manuální nástroj byl rozšířen o prvky poloautomatické anotace v rámci metodologie S-CREAM (Handschuh a kol., 2002). Sémantickou anotaci stránek tohoto typu však provádí minimum autorů, a to kvůli relativní složitosti pořizování anotací a nejasnému nebo nepřímému přínosu.

Technicky podobná je sémantická anotace webových stránek za účelem podpory činnosti internetových vyhledávačů. Tato oblast (Search Engine Optimization, SEO) je poměrně široká, my se zde zaměříme pouze na její anotační aspekt. Existují anotace standardní, jako jsou HTML META značky a sémantická klasifikace Dublin Core[6], které dokáže interpretovat většina vyhledávačů. Ty většinou vypovídají o obsahu dokumentu, např. o tématu, klíčových slovech, jazyce, autorovi apod. Většina vyhledávačů však přiznává tomuto druhu anotací poměrně malou váhu, protože se tato metadata (anotace) často zneužívají (např. dokumenty lákají na klíčová slova, která nemají s obsahem nic společného).

Druhou významnou skupinou anotací SEO jsou anotace vkládané do dokumentů speciálně pro konkrétní vyhledávač. Řada vyhledávačů totiž provozuje portály, jež umožňují například strukturované vyhledávání v produktových nabídkách různých internetových obchodů, srovnání cen apod. Generované produktové katalogy internetových obchodů proto často obsahují značky (návěstí), které vyznačují některé důležité položky (např. název, cenu, popis produktu) odpovídající slovníkům (ontologiím), jako jsou *GoodRelations* (viz kapitola 2) nebo *Schema.org*. Internetové obchody takto snadno umožňují robotům daného vyhledávače, aby při procházení svých stránek naplnily produktovými nabídkami databázi portálu vyhledávacího serveru (jde vlastně o jakousi symbiózu). Pozoruhodné je, že takto získaná data, spolu s jejich umístěním ve zdrojových stránkách, mohou vyhledávačům dobře posloužit také jako trénovací data pro automatické extrakční metody. Vyhledávač tak může pokrýt (s určitou přesností) relevantní zbytek internetu, který žádné speciální značky neobsahuje. Jde o zvláštní případ tzv. *bootstrappingu* popsaného v odst. 6.4.5.

Dalším rychle se rozšiřujícím případem sémantické tvorby dokumentů je přidávání obsahu na tzv. *sémantické wiki*. Jednotliví uživatelé zde anotují své příspěvky pomocí sdílené (např. firemní, univerzitní) ontologie a odměnou je jim lepší vyhledávání na wiki, kterému nečiní problém synonyma a zkratky, dokáže lépe pracovat s podobností témat a také s popularitou dokumentů.

6.2.2 Sekundární ruční anotace dokumentů

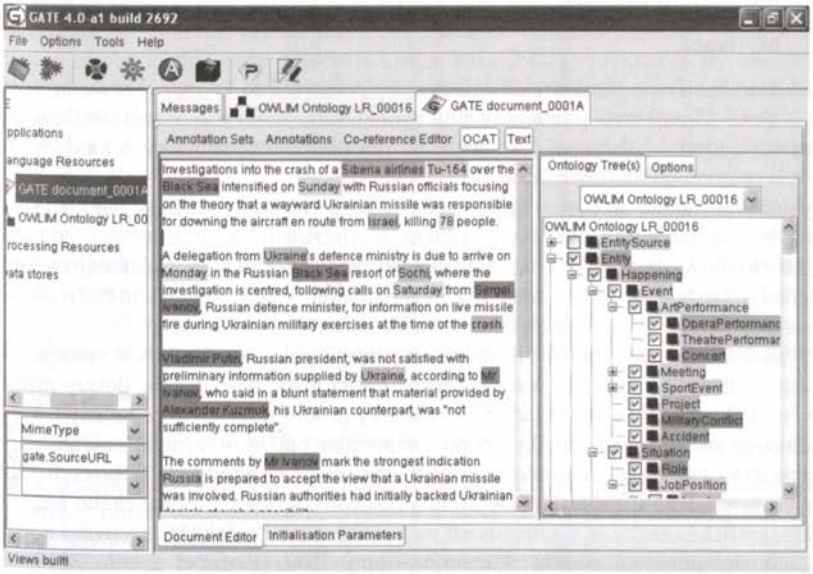
Sekundární anotací rozumíme anotování dokumentů nějakou dobu po jejich vzniku a většinou někým jiným než samotným autorem. Jak bylo řečeno v odst. 6.1.3.2, odpovídá tento způsob knihovnické anotaci obvykle prováděné za účelem vyhledávání. V tomto pojetí může jít například o anotaci klíčových slov nebo významných částí dokumentu. Jinou formou sekundární anotace je formalizace dokumentu psaného volným jazykem do předdefinovaných formulářů. Proces vyplňování „kolonek" formuláře je totiž analogický anotování příslušných částí dokumentu jménem příslušné kolonky. Nejčastějším případem tohoto druhu je formalizace medicínských zpráv ve zdravotnictví. Zde je samozřejmě příležitost pro (polo)automatické metody; intenzivní výzkum probíhá dokonce

[6] Sémantická klasifikace Dublin Core: http://dublincore.org (19. 12. 2010).

i v oblasti přímého převodu mluvených slov do příslušného vyplněného formuláře (Bisani a kol., 2008).

6.2.3 Ruční anotace trénovacích dat

Automatické extrakční nástroje, založené na učících se algoritmech (viz oddíl 6.4), potřebují pro svůj trénink velké množství příkladů správně anotovaných dokumentů. Tyto trénovací dokumenty nejčastěji pořizují lidští anotátoři. Existuje také mnoho uživatelských rozhraní, která tuto úlohu usnadňují. Anotační rozhraní bývají součástí vývojových prostředí podporujících celý proces extrakce. Nejrozšířenější z nich je GATE[7] (Cunningham a kol., 2002) disponující editory „Ontology Annotation Tool" a „Relation Annotation Tool". Na obr. 6.1 vidíme vpravo hierarchii konceptů ontologie o událostech, jimiž je anotován dokument v centrální části obrazovky. Podobně lze vyznačit relace mezi instancemi těchto konceptů. Méně známý systém Ellogon (Petasis a kol., 2002) analogicky jako GATE rovněž disponuje rozhraním pro anotaci dokumentů koncepty a relacemi z ontologií.



Obr. 6.1 Anotační rozhraní „Ontology Annotation Tool" jako součást GATE.

Většina anotačních nástrojů, včetně obou zmíněných, nabízí nějakou formu „našeptávání" na základě již provedených anotací. Některé z nich podporují funkci tzv. aktivního učení (active learning) popsaného dále v odst. 6.4.4, která nejenže „našeptává" jednotlivé anotace, ale také vybírá pro další anotaci vždy ten dokument, na němž si je systém svým „našeptáváním" nejméně jistý. Anotace se tak stává značně interaktivní a trénovací proces je rychlejší. Příkladem takového rozhraní je systém AKTive Media

[7] GATE systém pro zpracování přirozeného jazyka: http://gate.ac.uk (19. 12. 2010).

(Chakravarthy a kol., 2006), který navíc podporuje anotaci obrázků a jejich částí (např. anotaci postav na obrázku jejich jmény a přiřazení k existujícím anotacím v doprovodném textu).

6.3 Automatická extrakce informací: typy úloh a evaluace

V této části popíšeme nejčastější motivace pro automatickou extrakci informací („information extraction" – IE) a uvedeme několik členění extrakčních metod podle různých kritérií. Automatickou extrakcí informací budeme rozumět automatickou identifikaci údajů určité sémantické kategorie v analyzovaných dokumentech. Sémantické anotování potom spočívá v označení textu těchto dokumentů extrahovanými údaji. Pojmy „údaj" a „dokument" můžeme chápat velmi široce; údajem mohou být jak izolované výskyty například vlastních jmen v textu, tak složité hierarchické popisy událostí určitého druhu. Dokumentem můžeme rozumět jak čistý text, tak bohatě formátovanou webovou stránku.

6.3.1 Motivace

Jako hlavní motivace pro extrakci informací bývá uváděno tzv. *strukturované vyhledávání*. Oproti klasickému plnotextovému vyhledávání zná vyhledávací stroj nejen text dokumentů, ale také zařazení částí textu do relevantních sémantických kategorií (např. název či cena produktu, jméno člověka). To umožňuje:

- Vyhledávat zadaný text jen v rámci zadané kategorie; např. najít „jaguár" jen s významem zvířete nebo „Johnson" jen s významem „prezident" nebo „atlet".
- Klást složitější dotazy podobné dotazům do databází SQL; například najít v českých obchodech nabídky bicyklů, které mají vybrané komponenty a cenu do zadané výše.
- Porovnávat výsledky složitých dotazů napříč pokrytými dokumenty.

Výsledná aplikace nemusí mít nutně charakter vyhledávacího stroje vracejícího dokumenty setříděné podle relevance k dotazu, nýbrž může na položené dotazy přímo *odpovídat*. Aplikace může například vracet nabídky produktů vyhovující zadaným kritériím (s odkazem na zdrojové stránky) nebo zodpovídat otázky obecného charakteru. Tato doména bývá nazývána zodpovídání otázek a zaměřuje se na ni řada webových aplikací, jako je www.ask.com nebo www.wolframalpha.com. Kvalita automatického odpovídání na faktografické otázky je v současnosti na vysoké urovni a dokáže pracovat i v reálném čase, jak dokumentuje systém *Watson* vyvinutý IBM (Ferrucci a kol., 2010), který úspěšně soupeří s nejlepšími účastníky televizní vědomostní soutěže *Jeopardy* (obdoba české soutěže *Riskuj*).

Za jednoduchou, ale praktickou aplikaci IE lze rovněž považovat pouhé *vizuální* zvýraznění extrahovaných položek ve zdrojových dokumentech. Dokumenty takto anotované relevantními informacemi mohou výrazně urychlit čtení dokumentů lidmi za účelem například kontroly těchto informací. V rámci projektu MedIEQ (Karkaletsis a kol., 2008) byla IE využita pro poloautomatickou podporu posuzování kvality webových stránek. Jednalo se o zjištění přítomnosti formálních kritérií webových stránek vyžadovaných v rámci EU pro medicínské organizace. Mezi takovými kritérii jsou například jména a kontakty odpovědných osob, nepřítomnost reklamy, varování o nenahraditelnosti fy-

zického vyšetření lékařem apod. Úloha zde byla řešena jak po linii sémantického anotování (zvýraznění informací, na které se má soustředit lidský uživatel), tak po linii extrakce informací (dlouhodobé uložení klíčových informací, které byly následně opakovaně porovnávány s aktuálním stavem stránek).

Důležitou aplikací je rovněž podpora *strojového překladu*; extrakce informací zde může sloužit mimo jiné pro identifikaci nepřekládaných částí textu, jako jsou vlastní jména. Úkolem IE je zde tedy například zabránit překladu jména „Jan Kovář" na „John Smith."

Metody IE nacházejí uplatnění také při tvorbě *dialogových systémů*; tj. aplikací, jejichž cílem je vést s uživatelem dialog podobný přirozenému lidskému dialogu. Dialogové systémy mohou s uživatelem komunikovat prostřednictvím klávesnice a displeje; do této skupiny patří potomci historické Elizy (Weizenbaum, 1966), kteří se dnes nazývají *chatboti* (chatovací roboti). Naproti tomu hlasové dialogové systémy používají pro komunikaci syntetizovaný hlas a rozpoznávání řeči a uplatňují se při automatizaci call-center a v hlasovém ovládání zejména automobilů nebo mobilních telefonů. Metod IE se v obou případech používá pro extrakci položek uživatelova požadavku, například při dotazu na cenu letenky je třeba postupně získat údaje typu „odkud kam", čas apod.

Dalekosáhlejší motivací pro IE může být i alespoň částečné naplnění vizionářských představ o sémantickém webu (Berners-Lee a kol., 2001), ve kterém by automatičtí *agenti* sami „četli" webové stránky a ve prospěch svého majitele plnili vybrané úkoly, jako je například rezervace služební cesty či shánění výhodné dovolené. Za účelem „porozumění" službám nabízeným na internetu by tito agenti museli být schopni interpretovat strojově čitelné sémantické anotace u jednotlivých služeb, které by v masovém měřítku opět mohly pocházet od nástrojů IE. Studie (Hepp a kol., 2006) však na příkladu hledání ubytování v Rakousku dokumentuje, že na internetu často chybějí informace umožňující objektivní automatické porovnání jednotlivých nabídek. Průzkum mimo jiné ukázal, že ve zkoumaném vzorku mělo vlastní webové sídlo pouze 60 % poskytovatelů ubytování a z těchto webových sídel více než 30 % postrádalo klíčové informace o umístění a cenách. Aktuální informace o volných pokojích nabízelo jen 7 % poskytovatelů ubytování.

6.3.2 Kritéria pro členění úloh extrakce informací

Výhodou automatické extrakce informací je především rychlé pokrytí velkého počtu dokumentů. Nutnost automatické IE vyplývá z toho, že pouze minimální počet autorů webových stránek vybavuje své stránky nějakou formou sémantické anotace (výjimkou je v současnosti zřejmě jen FOAF a do jisté míry produktové katalogy). Nevýhodou automatické IE je však inherentní nepřesnost vlastní většině automatických metod; spolehlivost záleží na obtížnosti a omezenosti konkrétní úlohy, dostupnosti *extrakčních indicií* a volbě vhodných algoritmů a nástrojů. Automatická IE se opírá o různé typy extrakčních indicií, které pomáhají identifikovat výskyty relevantní informace v dokumentech. Některé typy takových indicií jsou uvedeny v tab. 6.1, čerpající mimo jiné z práce (Cohen, 2004). Metody extrakce informací lze členit podle celé řady kritérií. Uvedená kritéria by měla poskytnout představu o vysoké heterogenitě extrakčních úloh.

Tabulka 6.1 Některá kritéria pro členění úloh extrakce informací

Kritérium	Možné hodnoty kritéria	Příklady extrakčních metod a aplikací
A. Druh vstupních dokumentů	rozsáhlá množina dokumentů	Kromě extrakce je navíc třeba vyhledat relevantní dokumenty; metoda IE musí být robustní vůči zašuměným a irelevantním vstupům.
	jedno webové sídlo	Součástí úlohy IE je navigace po jednotlivých webových stránkách; typické pro tzv. wrappery.
	jeden dokument	Úkolem je nalézt výskyty relevantní informace uvnitř dokumentu, ignorovat nerelevantní části.
	fragmenty textu	Zpracovávají se pouze relevantní texty uvnitř domény; např. segmentace volně zadané poštovní adresy do částí (Borkar a kol., 2001) nebo extrakce hodnot vlastností produktu z textového popisu.
	text zatížený nejistotou, více variant textu (n-best)	Například extrakce místa a času textu pocházejícího od rozpoznávače řeči v automatickém dialogovém telefonním systému.
B. Množství formátování a gramatická správnost vstupního textu	čistý text	Metody využívající povrchové textové a hrubé lingvistické indicie jako identitu slov, slovní druhy. Například algoritmus LP^2.
	odstavce gramatických vět	Možnost využití lingvistických indicií pro „hlubší porozumění" textu, např. větných parterů.
	formátovaný text; např. HTML	Metody navíc využívají formátovací indicie, mohou se dokonce spoléhat výhradně na formátování (wrappery).
C. Obecnost metody (pokrytí různých druhů vstupu)	omezeno na homogenní skupinu dokumentů	Například jedno webové sídlo, kde stránky sdílejí formátování. Aplikace IE wrapperů.
	omezeno na doménu	Například extrakce údajů z nabídek práce nebo z oznámení o seminářích z různých zdrojů.
	bez omezení	Například extrakce relací různých druhů z libovolného textu bez ohledu na gramatičnost a množství formátování (tzv. otevřená IE; oddíl 4.5).
D. Složitost oboru hodnot extrahovaných prvků	konečný a „malý" obor	Například extrakce jmen kontinentů, zemí.
	odpovídá jednoduché formální gramatice	Například využití regulárních výrazů pro extrakci e-mailů a telefonních čísel.
	možno pokrýt složitějším vzorem	Například standardní poštovní adresa určité země.
	otevřené obory hodnot	Například extrakce jmen lidí z různých zemí.
E. Jednoznačnost hodnot extrahovaných prvků	relativně jednoznačné	Extrahované hodnoty představují hlavní indicii.
	nejednoznačné	Vysoké využití kontextu (okolní extrahované položky, slova, věty, dokumenty).
F. Struktura extrahovaných prvků	extrakce izolovaných prvků	Extrakce pojmenovaných entit (named entities); např. jmen lidí, měst, produktů, tel. čísel.
	extrakce prvků v binárních relacích	Extrakce entit a relací mezi nimi; např. člověk zaměstnán ve firmě.
	extrakce záznamů o mnoha prvcích	Plnění šablon; např. informací o akvizici společností, o teroristickém útoku (kdy, kdo, kde), o nákupu letenky (kdy, odkud, kam, ...).

G. Způsob zadání extrahovaných kategorií	předem zadané extrakční kategorie	Extrakce dané množiny pojmenovaných entit a relací (např. jméno člověka, firmy, zaměstnán-v).
	automatická extrakce nových relevantních sémantických kategorií	Otevřená extrakce informací, kde určení kategorií extrahovaných entit a relací je součástí extrakčního procesu.
H. Typ použitých extrakčních indicií	textový obsah a kontext extrahovaných prvků	Většina IE metod je založena na hledání častých obsahů nebo kontextů extrahovaných položek, ať už natrénovaných nebo ručně zadaných.
	pravidelné formátování v rámci předem určené skupiny dokumentů	Takzvané wrappery pro extrakci z konkrétních webových sídel, založené na neměnném formátování v okolí extrahovaných položek.
I. Původ použitých extrakčních indicií	ruční zadání	Využití regulárních výrazů a slovníků; např. jazyk *JAPE* nebo extrakční ontologie.
	trénovaný model konkrétního supervizovaného algoritmu	Natrénovaný model odvozený z typicky ručně označkovaných trénovacích dat, např. algoritmy LP^2, HMM, MEMM, CRF, SVM, bayesovské sítě.
	model vytvořený nesupervizovaným algoritmem na základě iniciálních příkladů	Systémy využívající bootstrapping, např. *Knowitall*, *Armadillo* nebo *Pankow*, postupně rozšiřují malou skupinu iniciálních příkladů.

6.3.3 Měření úspěšnosti extrakce informací

Úspěšnost extrakčních metod se nejčastěji měří pomocí dvou čísel: přesnosti a úplnosti. *Přesnost* (*P-precision*) vyjadřuje podíl správně extrahovaných položek (*SE*) mezi všemi extrahovanými položkami (*E*). Oproti tomu *úplnost* (*R-recall*) vyjadřuje podíl skutečně (správně) extrahovaných položek mezi všemi položkami, které měly být správně extrahovány (*S*).

Ideální extrakční systém by dosahoval 100% přesnosti (extrahované položky jsou vždy správné) a zároveň 100% úplnosti (systém extrahuje všechny relevantní položky). Při hodnocení úspěšnosti určitého systému je třeba vždy posuzovat obě statistiky, neboť mezi nimi existuje inverzní vztah. Je například jednoduché sestavit 100% přesný systém s nízkou úplností tak, že extrahuje pouze zřejmé případy, a naopak 100% úplný systém extrahující téměř vše, který má však nízkou přesnost. Pro účely srovnání různých systémů pomocí jediného čísla se proto často přesnost a úplnost kombinují do tzv. F-míry, která se počítá jako jejich harmonický průměr.

Přesnost a úplnost extrakce informací jsou dány vztahy

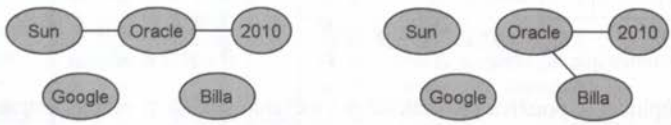
$$P = \frac{SE}{E}, \quad R = \frac{SE}{S}, \quad F = \frac{2PR}{P+R}. \tag{6.1}$$

Přesnost a úplnost se používá jako základ pro měření úspěšnosti většiny extrakčních úloh zmíněných v oddílech 6.4 a 6.5. Příkladem jejich použití je extrakce pojmenovaných entit, kde porovnávanými položkami jsou „správné" a automaticky extrahované výskyty pojmenovaných entit. K počtu správně extrahovaných položek *SE* přispívá každá přesně extrahovaná položka hodnotou jedna. Pro určité aplikace lze v tomto čísle vzít v úvahu i extrakce, které nejsou zcela přesné, ale stále ještě zvyšují kvalitu extrakce (např. extrakce položky o 10 správných slovech namísto 11 slov).

Při posuzování kvality extrakce pro celou kolekci dokumentů lze sumarizovat přesnost, úplnost i F-míru dvěma hlavními způsoby: makroprůměrováním a mikroprůměrováním. Při *makroprůměrování* se všechny tři statistiky počítají z kumulativních počtů *SE*, *E* a *S* v celé kolekci dokumentů tak, jakoby se jednalo o jediný velký dokument. Při *mikroprůměrování* jsou statistiky spočítány pro každý dokument zvlášť a v druhém kroku kombinovány, např. aritmetickým průměrem, který lze dále vážit dle významu jednotlivých dokumentů.

Analogie přesnosti, úplnosti a F-míry lze použít i pro měření správnosti seskupování extrahovaných pojmenovaných entit do celků (šablon). Jedna varianta takových měr se nazývá *Vilainova přesnost*, *Vilainova úplnost* a obdobou F-míry je *Vilainovo skóre* (Vilain a kol., 1996). Tyto míry se počítají na základě počtů imaginárních „čar", kterými by bylo nutné propojit jednotlivé pojmenované entity tak, aby každé dvě entity patřící k sobě byly mezi sebou vždy přímo anebo nepřímo propojeny. Snadno nahlédneme, že k takovému propojení *N* entit tvořících záznam postačí $N - 1$ čar. Není přitom podstatné, které entity jsou propojeny přímo a které nepřímo; relevantní je pouze počet těchto imaginárních čar v ručně anotovaných datech a počet chybějících a přebývajících čar v automatickém výstupu.

Vilainova přesnost i úplnost se počítají podle vzorců uvedených v rovnici 6.1 z počtů těchto imaginárních čar, přičemž *S* reprezentuje počet čar nutných k propojení originálního záznamu, který měl být extrahován, a *E* reprezentuje počet čar nutných k propojení extrahovaného záznamu. Pro výpočet Vilainovy úplnosti platí, že *SE* je počet čar nutných pro propojení *obrazu originálního záznamu* ve výstupu automatické extrakce. Obrazem záznamu rozumíme množinu automaticky extrahovaných pojmenovaných entit, které odpovídají položkám tvořícím originální záznam. Obraz záznamu může být v případě chyb extrakce rozdělen na několik nepropojených segmentů. V případě, že se některé pojmenované entity tvořící originální záznam nepodařilo extrahovat, lze s nimi pro účely výpočtu Vilainova skóre zacházet jako se samostatně extrahovanými entitami. Obrázek 6.2 ukazuje příklad extrakce relací typu „akvizice společnosti jinou společností v určitém roce". Obrazem ručně anotované relace „Oracle koupil Sun v roce 2010" jsou tři odpovídající entity v automatickém výstupu; ty jsou však propojeny jen jednou čárou, proto bude Vilainova úplnost rovna 1/2 = 0,5. Vilainovu přesnost získáme analogicky tak, že za *SE* dosadíme počet čar nutných k propojení obrazu skutečně extrahovaného záznamu v originálních záznamech; v našem příkladě bude tedy i Vilainova přesnost rovna 1/2 = 0,5. Vilainovo skóre se počítá z Vilainovy přesnosti a úplnosti, spočítaných celkově přes všechny uvažované relace, stejně jako F-míra v rovnici 6.1.



Obr. 6.2 Příklad spojování pojmenovaných entit do relace „akvizice": vlevo ruční anotace, vpravo chybný výstup automatického nástroje.

6.4 Přehled metod pro automatickou extrakci informací

V tomto oddílu se pokusíme zmapovat část automatických metod pro extrakci informací z textu. Nejdříve popíšeme principy automatických extrakčních metod založených na *ručně zadaných* extrakčních indiciích. Poté uvedeme přehled metod využívajících *strojové učení*, kdy jsou extrakční indicie získávány automaticky z trénovacích dat. Samostatně popíšeme tzv. *wrapperový přístup* představující metody opírající se o pevné formátování zdrojových dokumentů (využívají se zejména pro hromadný a pravidelný sběr dat v rámci konkrétního webového sídla). Jako doplněk k metodám využívajícím strojové učení ještě upozorníme na přístupy tzv. *aktivního učení* a *bootstrappingu*.

Věnovat se budeme především extrakci *pojmenovaných entit* v širším sova smyslu, méně pak extrakci *relací* mezi těmito entitami. Rozsah kapitoly neumožňuje popsat do detailu jednotlivé extrakční metody, a proto zůstává tento text na přehledové úrovni. Tři vybrané metody pro extrakci pojmenovaných entit jsou do větších podrobností popsány v oddílu 6.5. Podrobnější přehled extrakčních metod uvádí publikace (Moens, 2006) a s důrazem na popis existujících systémů také práce (Chang a kol., 2006).

6.4.1 Metody založené na ručně zadaných pravidlech

Extrakční metody založené na člověkem zadaných extrakčních indiciích jsou nejstarší a sahají od jednoúčelových programů napsaných v nejrůznějších skriptovacích jazycích až po sofistikované systémy, ve kterých lze informace určené k extrakci modelovat, s nimi spojená extrakční pravidla ladit a měřit jejich úspěšnost, plánovat pravidelné spouštění extrakčních úloh apod.

6.4.1.1 Extrakce informací založená na ontologiích

Data extrahovaná pomocí extrakčních nástrojů je obvykle třeba uložit pro pozdější využití, ať již za účelem vyhledávání v originálních dokumentech nebo pro jiné další zpracování. Úložištěm se mohou stát běžné datové soubory, databáze přístupné prostřednictvím SQL nebo znalostní báze napojené na ontologie. Podobně jako v případě databáze musí ukládaná data vyhovovat jejímu relačnímu schématu a integritním omezením, tak v případě ontologií musí ukládané instance konceptů a relací odpovídat schématu ontologie a splňovat její omezení.

Některé nástroje pro vytváření, plnění a používání ontologií podporují extrakci informací nepřímo prostřednictvím dodatečných modulů; k tomuto účelu slouží například *OntoLT plugin*[8] pro editor *Protégé*[9]. *Ontotext KIM*[10] (Kiryakov a kol., 2004) je platforma pro automatickou anotaci textů pomocí konceptů a instancí z ontologií pokrývajících především pojmenované entity často se objevující v novinových článcích (jména lidí, společností, geografických objektů apod.). Nejjednodušší metodou poloautomatické anotace dokumentů prvky z ontologie je vybavení konceptů jejich instancí a relací mezi nimi textovými „názvy" (labels), kterých může být pro každý jazyk několik. Výskyty těchto známých „názvů" v textu se potom považují za výskyty daného ontologického prvku

[8] OntoLT plugin pro editor Protégé: http://olp.dfki.de/OntoLT/OntoLT.htm (12. 4. 2011).
[9] Protégé; editor a prostředí pro práci s ontologiemi: http://protege.stanford.edu (19. 12. 2010).
[10] KIM Platform: www.ontotext.com/kim (19. 12. 2010).

a jsou takto automaticky anotovány. Samozřejmě zde dochází k problémům s nedostatečným pokrytím a s možnou víceznačností názvů.

Extrakční ontologie (Embley a kol., 2002) a (Embley, 2004) jsou původně manuální metodou založenou na přidání extrakčních znalostí do ontologie, která původně modelovala pouze doménové (věcné) znalosti. Extrakčními znalostmi se zde myslí regulární výrazy pro extrahované hodnoty, omezení těchto hodnot, výčty apod. Extrahovány mohou být jak výskyty pojmenovaných entit, tak výskyty relací mezi nimi. Na tento princip navazující *rozšířené extrakční ontologie* jsou popsány dále v odst. 6.5.1.

Řada systémů využívajících ontologie v extrakčním procesu je založena čistě na statistických technikách a nikoli na manuálně zadaných extrakčních znalostech. Například autoři práce (Dill a kol., 2003) popisují systémy *Semtag* a *Seeker* usilující o anotaci rozsáhlé části internetu pomocí hierarchické ontologie, jež modelují události, témata, osobnosti apod. Pro anotaci byly využity statistické metody IE zaměřené mimo jiné na desambiguaci víceznačných slov. Cílem projektu bylo sémantické vyhledávání soustředěné na správné zacházení s víceznačností hledaných termínů.

6.4.1.2 Jazyky pro tvorbu extrakčních pravidel

Vedle obecně použitelných procedurálních jazyků, specializujících se na práci s textem, jako je například *Perl* (Practical ExtRaction Language), existuje několik deklarativních jazyků vyvinutých výhradně pro extrakci informací. Nejznámějším zástupcem je extrakční jazyk *JAPE*, který je součástí extrakční komponenty *ANNIE* zasazené do architektury pro zpracování textu *GATE* (Cunningham a kol., 2002). *JAPE* (Java Annotation Patterns Engine) umožňuje tvorbu extrakčních pravidel využívajících regulární výrazy a práci s kategoriemi slov a symbolů (např. slovní druhy). Stejný účel jako *JAPE* má i *EOL* (Extraction Ontology Language), deklarativní jazyk vyvinutý pro reprezentaci extrakčních znalostí v rozšířených extrakčních ontologiích (viz odst. 6.5.1).

Některé systémy pro vývoj tzv. wrapperů využívají speciálních deklarativních i procedurálních jazyků, popisujících typické kroky nutné k nalezení a extrakci požadovaných dat z jednoho webového sídla. Tyto systémy budou podrobněji popsány v odst. 6.4.3 o wrapperech.

6.4.2 Algoritmy strojového učení nevyžadující pravidelné formátování

Při aplikaci učících se algoritmů pro IE je třeba, s ohledem na konkrétní úlohu, zvolit jednak samotný učící se algoritmus a jednak reprezentaci dokumentu, která zvolenému algoritmu vyhovuje.

6.4.2.1 Algoritmy dle reprezentace naučených znalostí a principu fungování

Většinu metod strojového učení, popsaných v kapitole 10 o adaptivních přístupech k dobývání znalostí a dále například v pracích (Berka, 2001) a (Hastie a kol., 2009), lze použít i pro účely IE, a to pro identifikaci pojmenovaných entit i relací. K dispozici jsou rovněž volně dostupné implementace těchto algoritmů, viz např. (Witten, Frank, 2005). Podle způsobu reprezentace naučených znalostí lze algoritmy strojového učení členit do tří základních skupin. V dalším výkladu uvádíme pro každou skupinu několik příkladů algoritmů včetně odkazů na další zdroje informací. Společným cílem všech těchto metod je naučit se z omezeného vzorku trénovacích dat takový soubor znalostí (extrakční mo-

del), který bude v praxi extrahovat co nejpřesněji na dříve neviděných (testovacích) dokumentech.

Pravděpodobnostní metody reprezentují naučené extrakční znalosti v podobě pravděpodobnostních distribucí, které se při tréninku odhadují na základě počtů různých událostí v trénovacích datech (např. kolikrát se v trénovacím textu vyskytlo slovo „Most" a kolikrát bylo slovo „Most" klasifikováno jako „město"). Mezi nejpoužívanější metody zde patří HMM, MEMM, CRF a bayesovské sítě. Základní principy prvních tří zmíněných metod popíšeme v oddílu 6.5.

Subsymbolické metody rovněž reprezentují extrakční znalosti s důrazem na numerické „váhy", avšak v jiných než pravděpodobnostních strukturách. V rámci aplikací IE byly nasazeny zejména SVM a neuronové sítě. V případě SVM jde o učení parametrů lineární nadroviny rozdělující vícedimenzionální prostor příznaků, v případě neuronových sítí se trénují váhy spojení mezi neurony v navazujících vrstvách sítě. Podrobný popis obou metod najdeme v kapitolách 10 a 12.

Indukce pravidel z dat je pro člověka často nejintuitivnější přístup k IE. Cílem je naučit se z trénovacích dat optimální soubor pravidel, která co nejpřesněji identifikují výskyty extrahovaných entit v dokumentech. Pravidla mají přitom tu výhodu, že jim, oproti předchozím dvěma přístupům, intuitivně porozumí i méně zkušený uživatel. Mezi pravidlové algoritmy IE patří například LP^2 dále popsaný v oddílu 6.5 nebo Rapier (Califf, Mooney, 1999). Oba přístupy jsou aplikací algoritmu pokrývání množin popsaného v kapitole 10.

6.4.2.2 *Algoritmy dle typu zpracovávaných dat*

Při extrakci pojmenovaných entit mají algoritmy strojového učení za úkol nacházet relevantní sousloví v analyzovaném textu a přiřazovat jim sémantické třídy. Příslušnost jednoho slova k určité sémantické třídě má přitom často silné implikace pro sémantické třídy okolních slov. Z tohoto důvodu je zajímavé ještě další hledisko, podle kterého lze algoritmy strojového učení pro IE rozdělit do dvou skupin.

Klasifikátory izolovaných příkladů jsou algoritmy, které rozhodují o sémantické příslušnosti každého posuzovaného příkladu *nezávisle na svých rozhodnutích o okolních příkladech*. Toto neznamená, že by při klasifikaci nebraly v úvahu kontext posuzovaného příkladu (např. okolní slova), ale pouze to, že každé klasifikační rozhodnutí je činěno nezávisle na okolních rozhodnutích. Výhodou tohoto přístupu je fakt, že existuje velké množství dobře fungujících klasifikačních algoritmů, které nebyly vyvinuty pro IE, nýbrž pro účely klasifikace nejrůznějších izolovaných příkladů. Nevýhodou naopak je, že tyto klasifikátory mohou na výstupu produkovat sekvence sémantických tříd, jejichž zřetězení je nepravděpodobné, nebo dokonce odporuje možným integritním omezením. V praxi jsou proto tyto algoritmy používány tak, že vracejí pro každý posuzovaný příklad více než jednu hypotézu. Jednotlivé hypotézy bývají často opatřeny určitou mírou věrohodnosti. V druhém průchodu je potom pro celou sekvenci posuzovaných příkladů nalezena nejlepší konzistentní cesta přes různé hypotézy doporučené pro jednotlivé příklady. Pro nalezení této cesty lze použít metod dynamického programování (Dasgupta a kol., 2006).

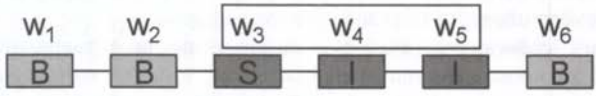
Značkovače sekvencí tvoří třídu algoritmů, které rozhodují o klasifikaci souvisejících příkladů společně. Jde většinou o metody specificky vyvinuté pro potřeby IE nebo jiných sekvenčních úloh, jako je značkování slovních druhů, rozpoznávání řeči nebo

optické rozpoznávání znaků (OCR). Značkovače sekvencí pracují nejčastěji na bázi zmíněných pravděpodobnostních metod. Jejich výhodou je právě společné rozhodování o celé posuzované sekvenci, čímž je eliminován integrační krok nutný v případě klasifikátorů izolovaných příkladů. To může vést jak k menší složitosti metody, tak k její vyšší přesnosti.

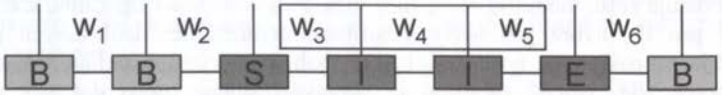
6.4.2.3 Možné reprezentace dokumentů

Algoritmy strojového učení uvažované v předchozí části rozhodují o klasifikaci příkladů. Pro účely extrakce pojmenovaných entit jsou tyto příklady nejčastěji reprezentovány těmito třemi způsoby.

Slova uspořádaná do sekvence odpovídající analyzovanému textu. Toto je nejpoužívanější přístup. Každý příklad může být reprezentován pomocí mnoha příznaků; například doslovná podoba posuzovaného slova (resp. tokenu – může jít například o číslo) a slov v jeho okolí, různé typy těchto slov (např. numerické, kapitalizované apod.), slovní druhy nebo členství těchto slov v seznamech známých pojmenovaných entit – gazetteerech (např. častá křestní jména, geografické názvy). Jako předpovídané sémantické značky slov se v praxi nejčastěji využívají *začátek* a *pokračování* textu náležícího do určité sémantické třídy (dále budeme takový text označovat jako výskyt třídy). To umožňuje jednak rozeznání několika následujících výskytů stejné třídy od jediného, jednak specializovaný trénink pro první slovo výskytu každé sémantické třídy, což může mít pozitivní vliv na přesnost díky tomu, že první slovo mívá často jiný charakteristický kontext i obsah než slova následující (např. křestní jméno jako součást jména člověka v kontextu věty). Další sémantickou značkou (obvykle majoritní) je *nezajímavé slovo*. Obrázek 6.3 ukazuje sekvenci šesti slov, z nichž třetí až páté tvoří pojmenovanou entitu, která je cílem extrakce. Pod každým slovem je jeho sémantická třída (význam tříd je uveden v legendě k obrázku).



Obr. 6.3 Reprezentace dokumentu jako sekvence slov. Třídy slov tvoří nezajímavé slovo (B, background), začátek pojmenované entity (S, start) a její pokračování (I, inner).

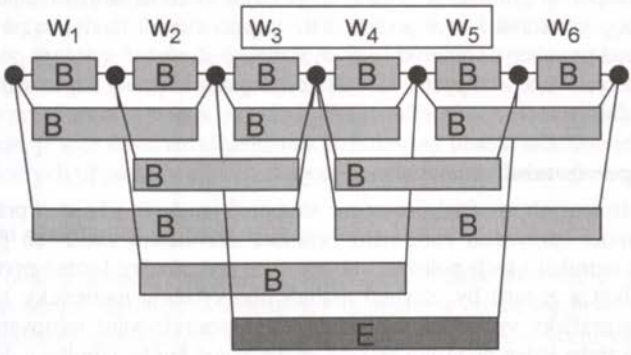


Obr. 6.4 Reprezentace dokumentu jako sekvence hranic mezi slovy. Třídy hranic tvoří nezajímavá hranice (B), začátek pojmenované entity (S), její pokračování (I) a konec (E, end).

Hranice mezi slovy uspořádané do sekvence v počtu o jedno větším, než je počet slov dokumentu. Příznaky jsou zde analogické jako v případě slov, pouze se vztahují ke slovům nalevo či napravo od hranice (Finn, 2006). Jako sémantické značky přiřazované hranicím při automatické klasifikaci lze použít *začátek*, *vnitřek* a *konec* každé extrahované třídy, viz obr. 6.4. Pro každou dvojici sémantických tříd, které mohou následovat bezprostředně po sobě, je třeba navíc definovat značky typu *konec první třídy* a *začátek*

druhé třídy. Analogicky jako v případě slov bývá majoritní značkou *nezajímavá hranice*. Reprezentace příkladů jako hranic mezi slovy je velmi podobná slovní reprezentaci. Její relativní nevýhodou může být větší počet značek, do nichž je třeba klasifikovat, a s tím související větší náročnost na velikost trénovacích dat.

Sousloví určitých délek uspořádaná do svazu definovaného nad slovy analyzovaného dokumentu, obr. 6.5. Pokud úloha dovoluje omezit délku extrahovaných segmentů slov, umožňuje tato metoda reprezentace použití příznaků definovaných na celém sousloví, o jehož klasifikaci se rozhoduje. To je výhoda oproti oběma předchozím způsobům reprezentace. Příznaky vztahující se k jednotlivým slovům nebo hranicím klasifikovaného sousloví přitom mohou být použity podobně jako v případě obou předchozích reprezentací. Využití většího počtu příznaků však většinou znamená větší nároky na velikost a kvalitu trénovacích dat. Jak již bylo popsáno pro případ klasifikátorů izolovaných příkladů, je vhodné pro jednotlivá sousloví získat skóre vyjadřující náležitost k určitému typu pojmenované entity a poté pomocí metod dynamického programování najít nejlepší cestu svazem sousloví. Pojmenované entity na takové cestě jsou potom výstupem extrakce. Tato metoda je využita v rozšířených extrakčních ontologiích popsaných v odst. 6.5.1.



Obr. 6.5 Reprezentace dokumentu jako svaz sousloví různých délek. Sémantické třídy tvoří pouze nazajímavá fráze (B) a pojmenovaná entita (E).

Klasifikátory izolovaných příkladů zmíněné v předchozí části se často využívají i při extrakci relací, kdy klasifikovaným příkladem bývá *n*-tice pojmenovaných entit, které by spolu mohly souviset. Například při extrakci relací typu „zaměstnán-v" mohou být tyto příklady tvořeny páry pojmenovaných entit typu „osoba" a „firma" které se vyskytují blízko sebe. Příznaky mohou pocházet z okolních slov, z adaptovaného výstupu syntaktických větných analyzátorů (parserů) apod. Vzhledem k provázanosti extrakce pojmenovaných entit a relací mezi nimi se jeví jako výhodné provádět obě úlohy zároveň (Roth, Yih, 2002) a (Getoor, Taskar, 2009) místo sekvenčního přístupu.

6.4.3 Wrappery

Významná část úloh extrakce informací z internetu se provádí z omezené skupiny zdrojových webových stránek, přičemž se lze s dostatečnou jistotou spolehnout na jejich ne-

měnné formátování. Takové úlohy lze relativně snadno řešit pomocí tzv. *wrapperů*. Wrapperem[11] rozumíme algoritmizovatelný postup opírající se o známé formátování dokumentů, které vede k identifikaci relevantních informací v rámci jednoho nebo více souvisejících dokumentů. Wrapper může být reprezentován různě, např. jako seznam pravidel, počítačový program nebo konečný automat. Wrappery lze rozdělit podle způsobu jejich vzniku na ručně vytvořené, supervizovaně trénované a nesupervizovaně trénované.

6.4.3.1 Ruční tvorba wrapperů

Jako ručně vytvořené wrappery lze nazvat i skripty vytvořené v obecných programovacích jazycích, které opakovaně provádějí stahování určitého webového obsahu na základě formátovacích prvků zdrojových stránek. Pro ruční tvorbu wrapperů ovšem existují vyspělé specializované nástroje, které ulehčují nejen tvorbu wrapperů, ale podporují i celý jejich „životní cyklus", tj. pravidelné spouštění, následné zpracování, ukládání a integraci extrahovaných dat apod. Zástupcem nástrojů open-source je systém *WebHarvest*[12], který umožňuje definici wrapperu pomocí speciálně vyvinutého deklarativního jazyka. Mezi komerční systémy patří například *Lixto* (Baumgartner a kol., 2001) a *Kapow*[13], které disponují grafickým rozhraním WYSIWYG umožňujícím „naklikat" extrahované položky a sestavit kroky potřebné k extrakci (login, navigace po jednotlivých stránkách, různá extrakce na různých typech stránek). Zmíněné systémy počítají primárně s manuální konstrukcí wrapperů, ale některé disponují i prvky supervizovaného učení popsanými v dalším textu.

6.4.3.2 Supervizovaně trénované wrappery

Cílem supervizovaných systémů pro učení wrapperů je ušetřit uživateli práci a wrapper definovat pomocí správného zobecnění několika uživatelem zadaných příkladů. Tak například pro extrakci všech položek určitého sloupce tabulky kromě první a poslední hodnoty (záhlaví a zápatí) by uživatel manuálního systému numericky zadal rozmezí řádků nebo je graficky vyznačil, zatímco uživatel supervizovaně trénovaného systému postupně zadá třeba jeden pozitivní příklad extrahované buňky tabulky a dále dva negativní příklady pro vynechání první a poslední buňky sloupce. Je zřejmé, že cílem supervizovaných wrapperů je minimalizovat interakci nezbytnou k přesnému naučení wrapperu, tedy potřebný počet pozitivních a negativních příkladů. Kromě dobrého učícího algoritmu je zde velmi významná i sada heuristik, jimiž se řídí generalizace ze známých příkladů; například kdy má systém po vyznačení buňky tabulky jako pozitivního příkladu navrhnout k extrakci celý inkriminovaný sloupec, řádek, popř. jejich úseky.

Metod a algoritmů pro tvorbu wrapperů existuje poměrně velké množství; dobrý přehled těchto technik nalezne čtenář v práci (Chang a kol., 2006). Naučený wrapper může být reprezentován například pomocí indukovaných pravidel odkazujících na kontext nebo obsah extrahovaných položek nebo pomocí konečného automatu (převodníku, transduceru) „absorbujícího" symboly vstupního dokumentu a vydávajícího extrakční rozhodnutí pro každý absorbovaný symbol. Mezi systémy využívající pravidla patří sys-

[11] Význam slova „wrapper" v softwarovém inženýrství je jiný než v extrakci informací; obvykle zde znamená „obálku" okolo komponenty SW zajišťující určité chování této komponenty nebo podporu určitého API.

[12] WebHarvest, systém pro manuální definici wrapperů: http://web-harvest.sourceforge.net (19. 12. 2010).

[13] Kapow Technologies: http://www.kapowtech.com (19. 12. 2010).

tém *WIEN* popsaný v práci (Kushmerick, 1997). Autor zde definuje různě mocné třídy wrapperů podle toho, jaké typy oddělovačů položek jsou schopné se naučit. Algoritmy *Whisk* (Soderland, 1999) a *Stalker* (Muslea a kol., 2001), využité mimo jiné pro extrakci informací o produktech v projektu *CROSSMARC* (Pazienza a kol., 2003), se učí pravidla podle principu pokrývání množin, popsaného v kapitole 10. Systém *Softmealy* (Hsu, 1998) je naopak představitelem těch wrapperů, které reprezentují naučenou znalost konečným automatem vytvořeným z omezeného množství trénovacích příkladů.

Zatím byla zmíněna pouze interaktivní tvorba wrapperů, existuje však řada aplikací i pro neinteraktivní přístupy. Lerman (Lerman a kol., 2003) popisuje algoritmy umožňující detekci změny formátování zdrojového webového sídla a automatické dotrénování (adaptaci) původních wrapperů vytvořených pro původní design stránek. Tyto algoritmy pracují na základě znalosti velkého množství pozitivních trénovacích příkladů, dostupných díky předchozí činnosti starých wrapperů na původním webovém sídle: extrahovaná data (např. názvy produktů, ceny) se totiž se změnou designu stránek obvykle nemění.

6.4.3.3 Nesupervizovaně trénované wrappery

Nesupervizovaně trénované algoritmy nevyžadují interaktivní učící proces s uživatelem, a dokonce nevyžadují ani žádné označkované trénovací příklady. Na vstupu mají pouze množinu stejně (podobně) formátovaných dokumentů ze stejného zdroje, jež však obsahují různá data. Systém *Roadrunner* (Crescenzi a kol., 2001) pracuje právě na základě předpokladu, že formátování ve vstupní množině dokumentů se (příliš) nemění a to, co se mění, jsou právě data určená k extrakci. Dynamický obsah dokumentů, extrahovaný tímto systémem, je sice často rozsáhlejší než požadovaný obsah, nicméně jeho relevantní část lze, v některých případech, získat relativně snadným následným zpracováním.

6.4.4 Aktivní učení

Aktivní učení (active learning) se snaží minimalizovat potřebu trénovacích dat při tvorbě extrakčního systému. Zatímco v základním přístupu je nutné nejprve nashromáždit dostatečné množství anotovaných trénovacích dat, při aktivním učení probíhá tvorba trénovacích dat interaktivně. Extrakční systém začíná s iniciálním nebo i prázdným extrakčním modelem (který nic neextrahuje) a s množinou neoznačkovaných dokumentů. V každém kroku učení systém vybírá takový ještě neanotovaný dokument, pro který si je „nejméně jistý" (v prvním kroku je pro anotaci vybrán libovolný dokument). Předpokladem je, že takový dokument, po ruční anotaci, nejlépe přispěje k vylepšení aktuálního extrakčního modelu. Poté, co uživatel ručně označkuje vybraný dokument, je tento dokument přidán k trénovacím datům a extrakční model je dotrénován. Toto dotrénování je nutné provést rychle, a proto se obvykle využívá forem trénovacích algoritmů umožňujících efektivní postupné přidávání nových trénovacích dat. Ruční anotace vybraného dokumentu se často usnadňuje tím, že jej existující extrakční model automaticky označkuje „tak jak umí". Na druhou stranu takto vždy vzniká riziko, že výsledná ručně opravená anotace se bude lišit od té, která by vznikla čistě manuálně (člověk má tendenci ponechat některá rozhodnutí extrakčního modelu, která by sám anotoval jinak). V momentě, kdy jsou nově vybrané dokumenty automaticky předanotovány s dostatečnou přesností a úplností, lze interaktivní trénink ukončit a začít natrénovaný model používat.

Mezi v praxi využitelné extrakční systémy aktivního učení patří nástroje Melita (Ciravegna a kol., 2003) a její následovník AKTiveMedia, jež umožňují navíc anotaci obrázků provázaných s textem (Chakravarthy a kol., 2006).

6.4.5 Bootstrapping

Bootstrapping[14] je technika nesupervizovaného nebo semisupervizovaného učení založená na principu analogie, která má stejný cíl jako aktivní učení – minimalizovat potřebu ručně anotovat velké množství trénovacích dat. Mezi první práce v tomto směru patří (Brin, 1998), kde byla navržena metoda *DIPRE* (Dual Iterative Pattern Relation Expansion, duální iterativní expanze relačních vzorů). Učící proces zde začíná tak, že uživatel opatří relativně malý vzorek správně anotovaných trénovacích dat. Může přitom jít jen o množinu pozitivních příkladů extrahovaných položek. Uvažujme například hypotetickou extrakční aplikaci snažící se na internetu najít veškeré názvy brodivých ptáků v češtině. Jako iniciální množinu známých exemplářů zadejme například „Čáp bílý", „Bukač velký" a „Kvakoš".

K učení je dále třeba rozsáhlá sada neanotovaných dokumentů, ve kterých se hledané entity přirozeně vyskytují (např. jako součást textu, v tabulkách apod.). V těchto dokumentech jsou nalezeny veškeré výskyty iniciálních pozitivních příkladů spolu s kontexty, ve kterých se vyskytují. Pro náš příklad to mohou být tyto kontexty nalezené internetovým vyhledávačem:

- „Čáp bílý je snad nejznámějším českým brodivým ptákem", „Čáp bílý je velký pták", „Čáp bílý je brodivý pták, který má".
- „Bukač velký je středně velkým druhem brodivého ptáka", „Bukač. Tento brodivý pták",
- „Kvakoš noční – brodivý pták nižší postavy".

Nalezené kontexty jsou podrobeny analýze, jejímž úkolem je vybrat a generalizovat kontexty charakteristické pro většinu doposud známých pozitivních příkladů. Tato analýza má různý charakter podle povahy dokumentů a extrahovaných položek. Pokud jsou dokumenty převážně v jednom jazyce a obsahují gramatický text, lze využít značkovačů POS a větných parserů pro daný jazyk a hledat podobnosti mezi naparsovanými strukturami. V případě vysoce strukturovaných dokumentů, obsahujících vysoké množství tabulek a jiných formátovacích prvků HTML, může jít naopak o nalezení takových pozic ve formátovací struktuře, které jsou typické pro většinu doposud známých exemplářů. V našem příkladě využijeme lemmatizátoru a značkovače POS, pomocí něhož zjistíme, že vzory „X je brodivý pták" a „X – brodivý pták" (kde X je rozvité podstatné jméno) se vyskytly hned několikrát; tyto vzory prohlásíme za charakteristické.

V dalším kroku jsou v neanotovaných dokumentech nalezeny všechny výskyty charakteristických kontextových vzorů a extrahovány jejich datové položky (označené X). V našem příkladě nalezneme frázi „je brodivý pták" pomocí internetového vyhledávače a pro každý výskyt extrahujeme rozvité podstatné jméno přímo předcházející uvedenému kontextu, pokud toto jméno pochází ze stejné věty. Uvedeným způsobem získáme k původním třem exemplářům ještě další: Volavka, Plameňák, Volavka červená, Šedá volav-

[14] Termín se v anglosaském prostředí obecně používá pro situace, kdy si nějaký subjekt „pomůže sám" v situaci, kdy se zdá, že se bez cizí pomoci neobejde. Doslova lze termín číst jako „zvednutí sebe sama za řemeny u bot".

ka, Euroasijský Curlew, Volavka popelavá, Velká modrá volavka, Koliha velká, Kolpík a Lesklý ibis.

Uvedený postup lze přitom několikrát opakovat se zvětšující se množinou pozitivních příkladů a získat tak větší pokrytí hodnot určených k extrakci. Podstatným rizikem je však zatažení falešně pozitivních příkladů při automatickém přidávání extrahovaných dat. Při iterativní aplikaci potom může nastat výrazný odklon od původního významu extrahované množiny hodnot (Brin, 1998). Techniky bránící „zašpinění" pozitivních příkladů spočívají jednak v evaluaci nově odvozených kontextových vzorů, jednak v evaluaci nově nalezených příkladů (Agichtein, Gravano, 2000). Jednoduchým opatřením jsou například požadavky na výskyt nově přidávaných vzorů i příkladů na několika různých webových sídlech, výskyt nového příkladu s více vzory apod. Můžeme také shlukovat stávající pozitivní příklady na základě podobnosti kontextů, ve kterých se příklady vyskytují, a vylučovat nové příklady příliš vzdálené od center těchto shluků. Poloautomatické metody potom spočívají v manuální inspekci nově přidávaných dat v každém kroku trénování.

Všimněme si, že uvedený postup lze použít jak k rozšíření množiny pozitivních příkladů nějakého konceptu (např. „brodiví ptáci"), tak pro extrakci relací typu „X vynalezl Y". Současné extrakční systémy využívající bootstrapping také pro extrakci relací jsou *KnowItAll* (Etzioni a kol., 2004) a *TextRunner* (Etzioni a kol., 2008). Jejich cílem je *otevřená* extrakce informací, kde typy extrahovaných položek nejsou dány předem, ale jsou výsledkem extrakčního procesu. Rozhraní systému TextRunner[15] umožňuje dotazování nad obecnými binárními relacemi extrahovanými z anglicky psaného internetu. Na podobném principu pracuje i systém Snowball (Agichtein, Gravano, 2000). Mezi další systémy využívající bootstrapping patří *Armadillo* (Ciravegna a kol., 2004) a *PANKOW* (Cimiano, Staab, 2004).

6.5 Vybrané metody pro extrakci informací

V tomto oddílu podrobněji popíšeme konkrétní příklady tří typů metod IE, reprezentujících přístupy popsané v předchozí kapitole. Cílem metody *extrakčních ontologií* je přímé „populování" ontologie z textu instancemi pojmenovaných entit a instancemi jednoduchých relací, přičemž se využívá jak manuálně zadaných extrakčních indicií, tak případných trénovacích dat a pravidelného formátování. Druhá skupina přístupů, učení *transformačních pravidel* (demonstrované na dvou příkladech), je založena na statistické indukci pravidel postupně přidávajících do anotovaného textu sémantické značky označující extrahované položky (pojmenované entity nebo relace). Poslední popsanou „rodinu" přístupů k IE tvoří tzv. pravděpodobnostní *grafické modely*, které patří mezi nejúspěšnější metody pro extrakci pojmenovaných entit i relací. V tomto textu popíšeme jejich aplikaci pro extrakci pojmenovaných entit.

6.5.1 Extrakční ontologie

Doménové ontologie (viz kapitola 1) slouží jako prostředek pro modelování omezené části světa, která je relevantní pro určitou aplikaci. Rovněž slouží k ukládání dat o této

[15] Systém *TextRunner* pro otevřenou extrakci relací: http://www.cs.washington.edu/research/textrunner (30. 12. 2010).

doméně; a to tak, že jsou *populovány* instancemi jednotlivých ontologických tříd a jejich relacemi (instance jsou obvykle ukládány do samostatných znalostních bází, které jsou s ontologiemi spojeny odkazy). Extrakce informací umožňuje efektivní automatické populování doménových ontologií. Pokud je však extrakční nástroj na ontologii nezávislý a nepoužívá ji během extrakce, je často třeba extrahované výsledky transformovat tak, aby se do ontologie daly vložit. Tento krok může být pracný a může vnést do výsledků chyby, například tehdy, když extrahovaná data nesplňují některá z integritních omezení vyžadovaných ontologií.

Proto se koncem 90. let minulého století vyvinul směr IE, ve kterém se extrakční nástroj přímo opírá o konkrétní doménovou ontologii, jež je upravená pro účely extrakce informací mimo jiné tak, že v sobě přímo obsahuje znalosti potřebné pro identifikaci instancí jednotlivých ontologických tříd. Termín *extrakční ontologie* zavedl D. Embley (Embley a kol., 2002), (Embley, 2004), jednalo se v podstatě o množinu jedné až několika tříd opatřených *datovými vlastnostmi*[16] (atributy), například třída *Auto* mající atributy *značka*, *rok výroby*, *najeté kilometry* a *cena*. Každý z atributů je v tomto přístupu doprovázen regulárními výrazy, které umožňují extrakci z textu. Tyto výrazy jsou z části určeny datovým typem atributu, oborem hodnot a jeho jednotkami a zčásti definovány uživatelem, jako například klíčová slova, v jejichž okolí se hodnoty atributu často vyskytují. Celá třída je dále vybavena omezeními, např. kardinalitními, která řídí seskupování nalezených hodnot atributů do instancí tříd. Extrakční ontologie byly využity především pro extrakci datových záznamů (např. o autech, fotoaparátech, telefonech) z webových stránek. Při extrakci je webová stránka nejdříve rozdělena na oblasti potenciálně obsahující záznamy (např. v případě tabulky jsou to její řádky) a v rámci každé oblasti se pak extrakční algoritmus snaží nalézt instance tříd, které by byly vzhledem k ontologii validní. Při změně domény je měněna pouze extrakční ontologie, ale nikoli extrakční algoritmy.

Autoři tohoto textu rozšířili původní metodu extrakčních ontologií tak, aby navíc dokázala zacházet s neurčitostí extrakčních znalostí, využít trénovací data (pokud jsou k dispozici), detekovat a využít pro extrakci pravidelné formátování v rámci dokumentu a zachytit složitější integritní omezení platná v rámci extrahovaných tříd (Labský, Svátek, 2008). Takto rozšířené extrakční ontologie jsou reprezentovány jazykem *EOL* („Extraction Ontology Language"). Extrakční ontologie zapsaná v jazyce *EOL* definuje jak jednotlivé *extrahovatelné třídy* a jejich *atributy* spolu s kardinalitními a datatypovými omezeními (viz obr. 6.6), tak i s nimi spojené *extrakční indicie*, mezi které patří zejména:

- Pravidla pro extrakci instancí tříd i hodnot atributů z textu opírající se o jejich typický obsah nebo kontext výskytu, využívají regulárních výrazů na slovech i na znacích. Využívají se rozsáhlé seznamy známých hodnot (např. častá křestní jména a příjmení pro extrakci jmen osob). Složitější pravidla lze definovat pomocí skriptovacího jazyka.
- Axiomy omezující vztahy mezi hodnotami atributů jedné instance třídy nebo obory hodnot atributu.
- Výstup algoritmů strojového učení trénovaných pomocí označkovaných dat.

Každá extrakční indicie může být spojena s odhady dvou pravděpodobností, popisujícími společný výskyt extrakční indicie a hodnoty předpovídaného atributu či třídy:

[16] Datová vlastnost přiřazuje objektu hodnotu jednoduchého datového typu, např. text, číslo, datum apod.

- Přesnost indicie odhaduje podmíněnou pravděpodobnost výskytu předpovídané entity za předpokladu pozorovaného výskytu indicie (můžeme také hovořit o míře postačitelnosti indicie).
- Úplnost indicie odhaduje, kolik procent výskytů předpovídané entity je doprovázeno také výskytem pozorované indicie (můžeme také hovořit o míře nutnosti indicie).

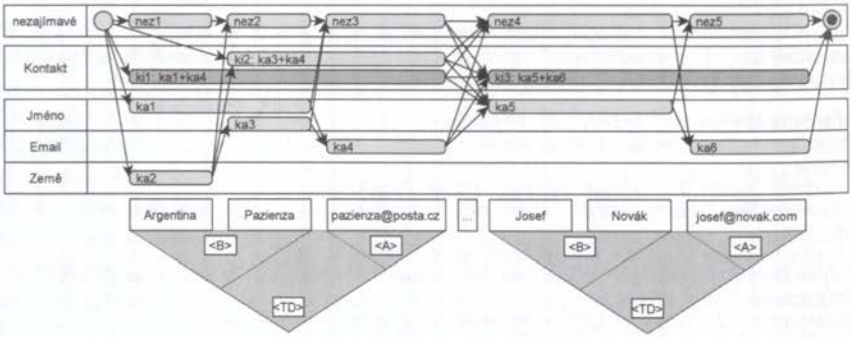
Při vlastní extrakci jsou v analyzovaném dokumentu nejprve nalezeny výskyty všech *extrakčních indicií* v ontologii definovaných. Z výskytů jednotlivých indicií jsou vytvořeni *kandidáti* na výskyt hodnot příslušných *atributů* (dále jen *KA*). Odhad pravděpodobnosti každého *KA* vzhledem k přítomnosti všech relevantních (pozorovaných i nepozorovaných) indicií je spočítán z přesností a úplností těchto indicií za předpokladu jejich *podmíněné nezávislosti* a známé apriorní pravděpodobnosti výskytu předpovídaného atributu v textu.

```
<class id="Kontakt">
 <script src="kontakt.js"/>
 <axiom cover="0.8"> jmenoOdpovidaEmailu($jmeno, $email) > 0 </axiom>
 <classifier id="c1" method="weka" classtype="attribute" name="weka.classifiers.rules.JRip"
  features="priznaky_jrip.feat" model="trenovany_model_jrip.bin" elements="*"/>

 <attribute id="jmeno" card="1" eng="0.80">
  <pattern id="inicial"> (A-Z) .? </pattern>
  <value>
   <pattern cover="0.5" p="0.8" ignore="lemma" case="CA|UC">
    <pattern src="krestni.txt"/> <pattern ref="inicial"/>? <pattern src="prijmeni.txt"/>
   </pattern>
   <pattern cover="0.25" p="0.4" ignore="lemma" case="CA|UC">
    <pattern src="krestni.txt"/> <pattern ref="inicial"/>? <tok type="alpha" case="CA|UC"/>
   </pattern>
   <pattern p="0.7" cover="0.5"> <phr label="c1.jmeno"/> </pattern>
   <pattern cover="0.3" type="format"> fits_in_parent </pattern>
   <length> <distribution min="1" max="6"/> </length>
   <axiom> jmenoJeValidni($) </axiom>
   <refers> jmenoKoreferuje($, $other) </refers>
  </value>
  <context> <pattern cover="0.1" p="0.6"> jméno :? $ </pattern> </context>
 </attribute>

 <attribute id="email" card="0-1" eng="0.60">
  <value> <pattern cover="0.9" p="0.9"> <pattern ref="generic.email"/> </pattern> </value>
 </attribute>
</class>
```

Obr. 6.6 Příklad rozšířené extrakční ontologie s jednou extrahovatelnou třídou a dvěma atributy.

Takto vytvoření *KA*, ohodnocení odhadem své podmíněné pravděpodobnosti, jsou vloženi jako uzly do grafu ve tvaru *svazu* procházejícího dokumentem (viz příklad extrakce kontaktních informací na obr. 6.7). Tento graf v dalších krocích poslouží k nalezení nejlepší cesty dokumentem, což bude cesta obsahující posloupnost uzlů s největším součinem pravděpodobností. Každý *KA* na nejlepší cestě svazem bude extrahován. Předtím jsou ale do tohoto grafu doplněny „nezajímavé" uzly (umožňující obejití každého *KA*), jejichž pravděpodobnost je odvozena od pravděpodobnosti nejpravděpodobnějšího obcházeného *KA*. Nejlepší cesta tímto svazem představuje předběžnou hypotézu o nejpravděpodobnějších extrahovatelných hodnotách atributů.

V dalším kroku se metoda pokusí nalézt ve zkoumaném dokumentu *pravidelnosti ve formátování*, které by mohly indikovat pravidelný výskyt hodnot určitých atributů. Mějme například (viz dolní část obr. 6.7) tabulku obsahující v prvním sloupci jména osob. Na základě indicií z extrakční ontologie mohlo dojít k odhalení většiny jmen, ale některá jména zachycena nebyla (např. část ženského jména „Argentina Pazienza" mohla být snadno zaměněna za název státu). Metoda proto hledá takové nové extrakční indicie opírající se o lokální formátování právě zpracovávaného dokumentu (ev. množiny dokumentů), které pro určitý typ atributu s vysokou přesností předpovídají výskyt *KA* tohoto atributu vyskytujícího se na (doposud) nejlepší cestě grafem. Indicie přesahující dolní prahy přesnosti a úplnosti (počítáno na základě počtu výskytů uvažovaného formátovacího vzoru a jím pokrytých *KA*) jsou pro daný dokument přidány do extrakční ontologie. Skóre uzlů v grafu jsou potom přepočítána a popřípadě jsou přidány uzly pro takto nově zjištěného *KA*.



Obr. 6.7 Svaz kandidátů na hodnoty atributů *jméno*, *e-mail* a *země* a kandidátů na instance třídy *Kontakt* sestávající z těchto atributů. *Nezajímavé* uzly v horní části vždy obcházejí určitého kandidáta na atribut. Nejlepší cestu dokumentem (sestává z uzlů s největším součinem pravděpodobností) tvoří dvě tmavě vyznačené instance třídy *Kontakt*: *ki1* a *ki3*. Formátovací struktura v dolní části obrázku může sloužit k odvození lokálně platných formátovacích indicií.

Předposlední krok extrakce spočívá v seskupování všech nalezených *KA* do *kandidátů na instance tříd* (*KI*). Tento proces je řízen *omezeními* z extrakční ontologie – zejména kardinalitními omezeními a axiomy určujícími přípustné nebo pravděpodobné kombinace hodnot atributů jedné instance třídy. Proces generování *KI* probíhá „zdola nahoru" – začíná s jednoprvkovými *KI* a seskupuje vždy toho nejpravděpodobnějšího *KI* s kompatibilními *KA* v jeho okolí. Okolí je zde definováno na formátovací úrovni – např. *KI* je nejdříve slučován s kompatibilními *KA* ve stejném formátovacím bloku, později o blok výše atd. Pravděpodobnostní skóre *KI* je dáno jednak pravděpodobnostmi *KA* v něm obsažených, jednak efektem indicií definovaných v rámci jeho třídy. Například nesplnění indicie typu „část jména osoby je v 80 % přítomna v její e-mailové adrese" sníží pravděpodobnostní skóre daného *KI*. Generování *KI* končí buď vyčerpáním prostoru možných seskupení, nebo dosažením jednoho z limitů (např. na počet *KI* nebo na pravděpodobnost nově generovaných *KI*).

V posledním kroku extrakce jsou *odstraněni* ti vygenerovaní *KI*, jež nejsou dle extrakční ontologie validní (např. nedosahují v některém atributu potřebné kardinality).

Poté jsou validní *KI*, přesahující určitý práh pravděpodobnostního skóre, *vloženi do grafu* tak, aby konkurovali samostatně stojícím *KA*. Na nejlepší cestě grafem tak mohou stát jak samostatné výskyty atributů (např. nezávislá jména osob a e-maily), tak instance tříd (např. k sobě patřící jméno a e-mail). Tendenci každého atributu tvořit třídy nebo vystupovat samostatně lze v exktrakční ontologii nastavit odhadem poměru počtu obou druhů výskytu daného atributu v analyzovaném textu.

6.5.2 Transformační pravidla pro extrakci informací

V tomto odstavci jsme pro podrobnější popis vybrali dva zástupce extrakčních metod založených na indukci pravidel: algoritmus LP^2 a sémantické parsování pomocí transformačních pravidel.

6.5.2.1 Algoritmus LP^2

LP^2 (Ciravegna, 2001) je příkladem algoritmu pro extrakci pojmenovaných entit, který reprezentuje naučenou extrakční znalost pomocí sekvence pravidel. Metoda vychází z tzv. *transformačního značkování* (Brill, 1994), které bylo původně vyvinuto pro úlohu určování slovních druhů („part-of-speech (POS) tagging"). Při průběhu značkování jsou transformační pravidla postupně aplikována na vstupní dokument tak, že do něj vkládají sémantické značky vyznačující segmenty určené k extrakci, nebo dříve vložené značky modifikují. Obecný algoritmus může pracovat se všemi reprezentacemi dokumentu popsanými v odst. 4.2.3; LP^2 však operuje nad hranicemi slov. Pravidla, která LP^2 používá, jsou trojího druhu: značkovací, kontextová a korekční.

Úkolem *značkovacího pravidla* je vložit do analyzovaného textu značku indikující začátek nebo konec určité pojmenované entity (na pravé straně pravidla je akce tuto značku vkládající). Na levé straně značkovacího pravidla je potom konjunkce testů na slova nacházející se v sousedství vkládané značky – konkrétně v okně M slov vlevo a M slov vpravo od místa vložení značky. Značkovací pravidla vkládají značky nezávisle na ostatních pravidlech a dříve vložených značkách. Nemusejí proto produkovat validní sekvenci značek, například po začátku pojmenované entity nemusí následovat její konec. Příklad pravidla vkládajícího počáteční značku „jména člověka" vidíme na obr. 6.8, kde značka je vkládána mezi sousedící slova w_{-1} a w_{+1}.

$$slovo(w_{-1}) = \text{„přednáší"} \;\&\; je_v_seznamu_křestních_jmen(w_{+1}) \rightarrow vlož \text{ <začátek_jména>}$$

Obr. 6.8 Příklad značkovacího pravidla.

Kontextová pravidla vypadají stejně jako značkovací pravidla, ale jejich aplikace je omezena jen na určitý kontext vzhledem ke značce dříve vložené jinými značkovacími nebo kontextovými pravidly. Příkladem je na obr. 6.9 vyobrazené pravidlo vkládající koncovou značku jména člověka v kontextu několika slov napravo od dříve vložené značky začátku jména člověka.

$$kontext = \text{max. } N \text{ slov vpravo od značky <začátek_jména>}$$
$$začíná_vel_písmenem(w_{-1}) \;\&\; nezačíná_vel_písmenem(w_{+1}) \rightarrow vlož \text{ <konec_jména>}$$

Obr. 6.9 Příklad kontextového pravidla.

Autoři LP^2 uvádějí, že nejčastějším druhem chyby vycházejícím z aplikace předchozích dvou typů pravidel jsou značky posunuté o jedno nebo několik málo slov od správného umístění. *Korekční pravidla* mají za úkol napravit tyto chyby tak, že dříve vloženou značku za určitých podmínek posunou (nevkládají nové značky). Stejně jako kontextová pravidla mohou na levé straně odkazovat na dříve vložené značky. Příklad korekčního pravidla uvedeného na obr. 6.10 posouvá značku „konec jména" o další slovo v případě, že je jméno následováno slovem „mladší".

značka_před(w_{-1}) = <konec_jména> & *slovo*(w_{-1}) = „mladší" → *posuň_značku* za w_{-1}

Obr. 6.10 Příklad korekčního pravidla.

Při běhu aplikuje algoritmus naučená pravidla v pořadí, ve kterém se je naučil, a to nejdříve značkovací, po nich kontextová a nakonec korekční. Výsledné značkování však nemusí být validní, proto jsou v posledním *validačním* kroku odstraněny takové počáteční a koncové značky, které se nevyskytují v páru.

Učení pravidel probíhá na základě trénovacího korpusu dokumentů, do něhož byly ručně vloženy správné značky. Učení probíhá „zdola nahoru" generalizací konkrétních pozitivních příkladů (značek) viděných v trénovacích datech. V prvním kroku se stává kandidátem na značkovací pravidlo každý pozitivní příklad značky pozorovaný v trénovacích datech. Slova v okně $\pm M$ slov okolo značky se dostávají na levou stranu pravidla a pravou stranu tvoří akce vkládající pozorovanou značku. Každý takový kandidát na pravidlo je dále generalizován postupným ubíráním podmínek z levé strany. Tím se dosahuje většího očekávaného pokrytí na neviděných testovacích datech. Kvalita každého uvažovaného pravidla se měří na „pracovním korpusu", který je shodný s trénovacím korpusem, ale chybí v něm trénovací značky. Kvalita pravidla je určena redukcí celkové chyby, které by bylo aplikací pravidla na pracovním korpusu dosaženo (typicky počet opravených chyb zmenšený o chyby způsobené). V každém kroku je vybrán nejlepší kandidát na pravidlo, učení tedy probíhá „hladově". Každé nové pravidlo je při svém naučení na pracovní korpus aplikováno a značky jím vyprodukované v pracovním korpusu zůstávají. Tím, že značky naučených pravidel v pracovním korpusu zůstávají, jsou při naučení nového pravidla z množiny uvažovaných příkladů efektivně odstraněny všechny příklady pravidlem pokryté; LP^2 je tedy příkladem algoritmu *pokrývání množin* (viz kap. 10). Učení probíhá analogicky pro všechny tři druhy pravidel v uvedeném pořadí.

Po indukci pravidel se provádí ještě prořezávání naučených pravidel za účelem odstranění redundantních pravidel (např. pravidel obsažených v jiných pravidlech). Podobně jako u jiných trénovaných metod lze indukci transformačních pravidel ukončit na základě poklesu přesnosti extrakce doposud natrénovaného modelu na anotovaném *validačním* vzorku dat, který se nepoužije pro trénování – jde o ochranu před *přetrénováním* („overfitting"), tj. přílišným přizpůsobením modelu specifickým vlastnostem trénovacích dat na úkor použitelnosti pro data neznámá.

6.5.2.2 Sémantické parsování pomocí transformačních pravidel

Metoda transformačních pravidel, popsaná v předchozí části, je dostatečně obecná a pro účely extrakce informací ji lze použít více způsoby. Juříček (Jurčíček a kol., 2009) popi-

suje metodu sémantického parsování aplikovanou na výstup rozpoznávače řeči v dialogovém telefonním systému s výsledky pro domény prodeje letenek a turistického informačního systému. Účelem metody je pro vstupní text, popisující požadavek uživatele, vyplnit jednu šablonu sestávající z typu požadované akce (cíl) a z argumentů této akce. Obrázek 6.11 ukazuje vyplněnou šablonu pro dotaz „Kolik stojí *nejlevnější* letenka z *Prahy* do *New Yorku* na *příští týden*". V tomto přístupu nepřiřazují transformační pravidla sémantické značky segmentům analyzovaného textu, ale místo toho přímo vyplňují pracovní verzi samotné výstupní šablony. Ta je na počátku extrakce předvyplněna apriori nejčastější hypotézou; např. *přepojit na operátora* (bez argumentů).

Cíl: dotaz na cenu letenky	Odkud: Praha	Kam: New York
	Kdy: příští týden	Typ: nejlevnější

Obr. 6.11 Příklad šablony s vyplněným cílem a čtyřmi atributy.

Učení i aplikace transformačních pravidel probíhá analogicky k již popsanému algoritmu LP^2. První rozdíl spočívá v možném obsahu obou stran pravidel. Na levé straně pravidel vystupuje konjunkce jednoho nebo více argumentů následujících tří typů: n-gramy slov ze zkoumané fráze (autoři konkrétně použili uni- až trigramy a „děravé" bigramy s možností vynechání 1–3 slov), současné hodnoty atributů v šabloně a současnou hodnotu cíle. Protože tyto příznaky dobře neumožňují zachytit vztahy mezi vzdálenějšími slovy, využívají autoři závislostní syntaktický parser k tomu, aby obohatili n-gramy slov pozorované v lineární větě o n-gramy slov spojených hranou v *závislostním odvozovacím stromu*. Na pravé straně pravidel jsou možné čtyři akce: *změň cíl* a *přidej*, *smaž* nebo *změň* hodnotu *jednoho atributu*. Výhodou tohoto přístupu je oproti LP^2 fakt, že při trénování nemusejí být v textu vyznačena umístění hodnot jednotlivých atributů ani cíle; extrahované hodnoty atributů i cíle mohou navíc nabývat i hodnot, které se v samotném zdrojovém textu nevyskytují. Je však třeba zdůraznit, že metoda byla úspěšně aplikována na krátké věty (byť zatížené chybou rozpoznávání řeči) a autoři nepopisují aplikaci na rozsáhlejší dokumenty.

6.5.3 Grafické pravděpodobnostní modely

Grafický pravděpodobnostní model obecně slouží pro modelování *složené pravděpodobnostní distribuce množiny náhodných veličin* za předpokladu *podmíněné nezávislosti* mezi některými z nich. V grafických modelech jsou náhodné veličiny reprezentovány jako uzly grafu. Hrany grafu mezi dvěma uzly značí, že model přímo modeluje závislost mezi veličinami spojenými touto hranou. Pokud hrana mezi dvěma veličinami chybí, jsou tyto veličiny modelovány jako podmíněně nezávislé. Část uzlů grafu představuje pozorované náhodné veličiny (typicky slova a jejich vlastnosti) a část modeluje skryté veličiny, jejichž hodnoty chceme vědět (typicky sémantické značky asociované s každým slovem). Grafické modely se dělí do dvou skupin.

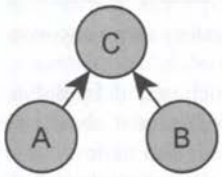
Orientované grafické modely modelují výslednou složenou pravděpodobnostní distribuci jako součin „menších" podmíněných distribucí obsahujících jen některé z modelovaných proměnných. Konkrétně jde o součin tolika podmíněných distribucí, jako je v grafu uzlů, přičemž distribuce pro každý uzel U má tento charakter

P(*hodnota náhodné veličiny uzlu U* | *hodnoty náhodných veličin všech rodičů uzlu U*).

Rodičem uzlu *U* zde rozumíme přímého předka, tj. uzel, ze kterého vede orientovaná hrana do *U*. Pokud uzel žádné rodiče nemá (nevedou do něj žádné hrany), je jeho náhodná veličina modelována apriori rozdělením

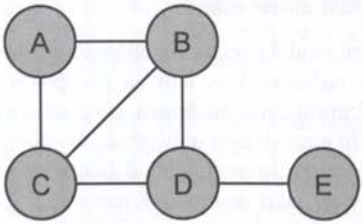
P(*hodnota náhodné veličiny uzlu U*).

Složenou distribuci $P(A, B, C)$ lze například rozložit, za předpokladu podmíněné nezávislosti veličin A a B při daném C, na tři faktory $P(A)$, $P(B)$, $P(C|A, B)$. Graf příslušného orientovaného modelu ukazuje obr. 6.12. Do rodiny orientovaných extrakčních modelů patří tzv. *HMM* a *MEMM*, které popíšeme v následujících sekcích.



Obr. 6.12 Příklad orientovaného grafického modelu.

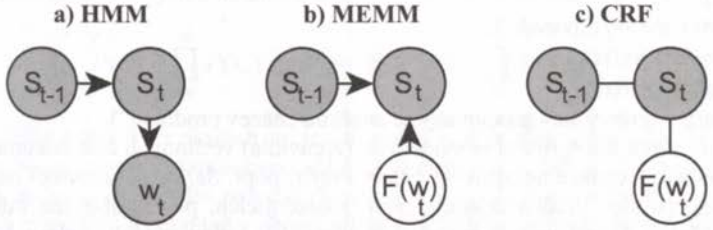
Neorientované grafické modely nemodelují složenou distribuci rozkladem na součin „menších" podmíněných distribucí, ale rozkládají ji na součin tzv. *potenciálových funkcí*. Potenciálová funkce je funkce vracející reálné nezáporné číslo. Je definována na náhodných veličinách tvořících tzv. *kliku* grafu náhodných veličin. Klikou grafu rozumíme každý *největší plně propojený* podgraf. Příklad grafu z obr. 6.13 ukazuje pět náhodných veličin tvořících tři kliky: *ABC*, *BCD*, *DE*. Sdružená distribuce modelovaná celým grafem je v neorientovaných grafických modelech počítána jako normalizovaný součin potenciálových funkcí, kterými je graf ohodnocen. Nejznámějším neorientovaným grafickým modelem s aplikací pro IE jsou tzv. *CRF*, popsané v dalším textu. Více o grafických modelech najdeme v pracích (Bishop, 2006) a (Koller, Friedman, 2009).



Obr. 6.13 Příklad neorientovaného grafického modelu o dvou klikách.

V následujícím textu popíšeme principy tří extrakčních metod vycházejících z grafických modelů. Ve všech případech jde o *značkovače sekvencí*, které pozorované sekvenci slov W přiřazují *nejpravděpodobnější sekvenci sémantických značek S*. Grafické modely pro tyto tři metody jsou zobrazeny na obr. 6.14. Náhodné veličiny S_t a S_{t-1} zná-

zoňují hodnotu sémantické značky aktuálního a předcházejícího slova v anotovaném dokumentu; w_t značí aktuální slovo na pozici t (se značkou S_t) a $F(w_t)$ představují množinu *příznaků* definovaných pro aktuální slovo s možným využitím celého vstupu. Jedním příznakem $f(w_t) \in F(w_t)$ je obvykle binární tvrzení o přítomnosti nějakého jevu v pozorovaném textu vzhledem k pozici aktuálního slova, např. „předchozí slovo je ‚sleva'" nebo „aktuální slovo je složeno z číslic". V některých případech je jako součást definice příznaků chápána také hodnota předchozího stavu a hodnota předpovídaného stavu, tedy například „aktuální slovo je složeno z číslic, předchozí stav je s' a předpovídaný stav pro aktuální slovo je s". Obrázek 6.14 ukazuje rozdíl přístupů k modelování směru závislostí náhodných veličin. Vybarvené uzly znázorňují veličiny generované modelem; prázdné uzly odpovídají pozorovaným veličinám. Na rozdíl od ostatních přístupů generativní HMM modeluje (generuje) i pravděpodobnosti pozorovaných slov.



Obr. 6.14 Modelování směru závislosti náhodných veličin v modelech HMM, MEMM a CRF.

6.5.3.1 *Skryté Markovovy modely*

Skrytý Markovův model (Hidden Markov Model, *HMM*) je prvním ze tří grafických pravděpodobnostních modelů, které v této kapitole popíšeme. Jde o orientovaný grafický model, jehož struktura je pro konkrétní příklad znázorněna na obr. 6.15. HMM je definován jako nedeterministický konečný stavový automat, v němž je každý stav doplněn o dvě pravděpodobnostní distribuce:

- $P(w|s)$ – pravděpodobnost generování slova w v současném stavu s (také lexikální, emisní nebo výstupní distribuce),
- $P(s|s')$ – pravděpodobnost přechodu ze stavu s' na další stav s (přechodová distribuce).

Pro zjednodušení notace předpokládejme, že model zahájí svou činnost vždy v jednom dedikovaném stavu s_{start}, který vždy generuje stejné iniciální slovo w_{start}. Alternativně lze definovat ještě iniciální distribuci určující pravděpodobnost startu z každého ze stavů. Podrobná definice HMM a souvisejících algoritmů je nad rámec tohoto textu a čtenáře odkazujeme na publikace (Manning, Schuetze, 2000), (Jelinek, 1997) a (Rabiner, 1989).

Vraťme se nyní krátce k obr. 6.14, části a), kde grafický model HMM zahrnuje tři náhodné veličiny: předchozí stav konečného automatu S_{t-1}, současný stav S_t a slovo w_t asociované se současným stavem. Orientace hran říká, že současný stav je modelován jako závislý na předchozím stavu a slovo je modelováno jako závislé na současném stavu. Slovo je tedy „generováno" současným stavem, proto je HMM označován jako „generativní model". Je třeba zdůraznit, že graf konečného automatu HMM (obr. 6.15) je něco jiného než obecné schéma grafického modelu HMM (obr. 6.14, část a). V grafu

konečného automatu uzly reprezentují stavy konkrétního automatu a přechody mezi nimi, zatímco grafický model je stejný pro všechny HMM.

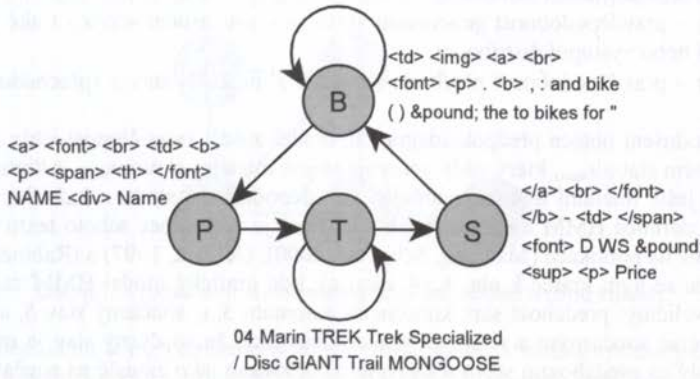
Modelování dokumentu pomocí HMM

Pro účely extrakce informací můžeme pomocí HMM modelovat celý text zkoumaných dokumentů. HMM je *generativním modelem textu* a základní představa je proto taková, že model náhodně (dle přechodových pravděpodobností) přechází ze stavu do stavu a v každém stavu generuje náhodně *jedno slovo* (dle lexikálních distribucí). Pro extrakci informací obvykle asociujeme některé stavy se *sémantickými značkami*. Strukturu modelu (počet stavů a jejich propojení) můžeme definovat ručně, nebo lze využít některé z metod učení struktury modelu, viz např. (Freitag, McCallum, 2000).

Uvedeme nyní jednoduchý příklad HMM pro extrakci výskytů jediného typu sémantické značky – názvu produktu (např. z katalogů internetových prodejců). Náš model bude mít pouze 4 stavy:

- *B* – Background („pozadí"),
- *P* – Prefix (prefix),
- *S* – Suffix (sufix),
- *T* – Target (cílový stav asociovaný se značkou „název produktu").

Úkolem stavu *Background* je modelovat (zpravidla) většinovou část dokumentu, která je z hlediska extrakce nezajímavá. Stavy *Prefix*, popř. *Suffix*, se soustředí na modelování typických slov předcházejících, popř. následujících, po extrahované informaci – jménu produktu. Stav *Target* je pro nás nejzajímavější, neboť modeluje slova, která tvoří jméno produktu, které má být extrahováno. Struktura tohoto modelu je znázorněna na příkladu námi zkoumané domény cyklistických produktů na obr. 6.15, který pro každý stav ukazuje několik jím generovaných nejpravděpodobnějších slov a formátovacích značek. Šipky značí nejpravděpodobnější přechody z každého stavu; ostatní přechody lze zakázat nebo jejich pravděpodobnosti ponechat na nízkých hodnotách vycházejících z trénovací procedury nebo z vyhlazení. Pravděpodobnosti nejsou pro přechodové ani lexikální distribuce kvůli stručnosti uvedeny.



Obr. 6.15 Struktura HMM pro extrakci názvů produktů a nejčastěji emitovaná slova.

Uvedený HMM by mohl generovat dokument například tak, že začne ve stavu *B*, 50krát zde setrvá (tj. vygeneruje 50 nezajímavých slov), poté přejde do stavu *P*, kde vygeneruje slovo „Name", dále přejde do *T*, kde stráví 3 cykly a postupně vygeneruje jméno produktu „Trek 04 Disc", poté přejde do *S*, generuje zde odřádkování „
", dále přejde zpět do *B* atd.

Konečný automat zde tedy pomyslně generuje jak sémantické značkování textu, tak (zprostředkovaně přes vygenerované značky) i text samotný. Předpoklad, že lze reálný zpracovávaný text chápat jako generovaný konečným automatem, je samozřejmě zjednodušující, avšak pro účely extrakce pojmenovaných entit dosahují HMM relativně dobrých výsledků. HMM tedy modeluje *sdruženou pravděpodobnostní distribuci* $P(W, S)$ definující pravděpodobnost (jakéhokoli) výskytu textového řetězce W s (jakýmkoli) označkováním S. Tuto distribuci přitom modeluje za pomoci rozkladu $P(W, S) = P(S)P(W|S)$. Zásadního zjednodušení se HMM dopouští tehdy, když aproximuje $P(S)$ a $P(W|S)$ pomocí zmíněných lexikálních a přechodových distribucí

$$P(S) = \prod_{t=1}^{N} P(s_t \mid s_{t-1}) \quad \text{a} \quad P(W \mid S) = \prod_{t=1}^{N} P(w_t \mid s_t), \tag{6.2}$$

kde N je délka textu W ve slovech, w_i je i-té slovo, s_i je značka i-tého slova a $s_0 = s_{\text{start}}$. *Markovovým omezením* (Markovovou vlastností) popsaného náhodného procesu se nazývá fakt, že budoucí chování procesu závisí pouze na jeho současném stavu, ale už ne na stavech minulých. Konkrétně v našem případě je budoucí sémantická značka modelována jako závislá pouze na současné sémantické značce, ale nikoli na značkách předchozích nebo na konkrétních slovech (v jakýchkoli pozicích). Každé slovo je dále modelováno jako závislé pouze na jeho vlastní sémantické značce, nikoli na značkách nebo slovech v okolí.

HMM generující symboly na stavech jsou jen jednou variantou HMM, stejně se můžeme setkat i s HMM generujícími symboly na přechodech. V takovém případě je emisní distribuce modelována jako $P(w|s', s)$. Obě varianty mají stejnou vyjadřovací sílu. Popsaná Markovova vlastnost omezila historii relevantní pro předpověď dalšího vývoje pravděpodobnostního procesu na současný stav s. V takovém případě jde o tzv. *Markovův model prvního řádu*, neboli o tzv. *bigramový model*, jehož přechodová distribuce předpovídá budoucí stav na základě jednoho současného stavu. Pro různé praktické úlohy má však smysl požadovat delší historii. Uvažujeme-li relevantní historii dvou stavů (*trigramový model*), změní se přechodová distribuce na $P(s|s'', s')$. Odpovídající konečný automat si představme tak, že každý stav odpovídá vždy dvojici značek (s'', s') a přechody z tohoto stavu vedou pouze do stavů, jejichž první značka je s', tedy do stavů typu (s', s). V trigramovém modelu můžeme emisní distribuce modelovat stejně jako v modelu bigramovém, tj. učinit generované slovo závislé pouze na současné značce. V takovém trigramovém modelu je stejná emisní distribuce $P(w|s)$ sdílena všemi stavy typu $(*, s)$. Alternativě může každý z těchto stavů mít vlastní emisní distribuci, čímž se generované slovo stává závislým na současném a předchozím stavu. Analogicky můžeme pokračovat modely vyšších řádů. Jak je vidět, s delší historií narůstá počet stavů modelu (a tím i počet parametrů) kvadraticky, a proto brzy narazíme na problém rozředění trénovacích dat a na praktickou nemožnost velký počet parametrů modelu spolehlivě odhadnout.

Extrakce informací pomocí HMM

Zatím jsme popsali pouze způsob, jak je dokument pomocí HMM modelován. Pro použití HMM k extrakci informací je třeba ještě vyřešit následující dvě hlavní úlohy:

- Při tréninku – z trénovacích dat natrénovat optimální přechodové a emisní pravděpodobnosti modelu.
- Při extrakci – pro zkoumaný dokument určit nejpravděpodobnější cestu (posloupnost stavů) natrénovaným modelem, kterou by tento dokument mohl být vygenerován. Jde tedy o to, při pozorované posloupnosti slov W, zjistit nejpravděpodobnější sekvenci značek S (viz rov. 6.3). Jelikož W a tedy i $P(W)$ je konstantní a nemá pro výběr maxima význam, stačí hledat sekvenci značek S maximalizující hodnotou sdružené distribuce $P(S', W)$. Tuto úlohu plní tzv. *Viterbiho algoritmus*.

Pro praktické použití HMM je podstatné, že obě úlohy lze řešit snadno a relativně efektivně. Proces trénování je nejjednodušší v těch případech, kdy můžeme v ručně označkovaných trénovacích datech *přímo pozorovat* sekvence stavů. V našem příkladu v trénovacích datech přímo pozorujeme jak cílové stavy T (neboť ručně vyznačená jména produktů jsou reprezentována právě jedním stavem T), tak i *prefixové* a *sufixové* stavy (neboť struktura modelu je taková, že model musí před jménem produktu generovat právě jedno prefixové slovo a po jménu produktu právě jedno sufixové slovo). Zbytek slov v trénovacích datech přísluší zbývajícímu stavu B.

V případě, že z trénovacích dat lze jednoznačně vyčíst sekvenci příslušných stavů HMM, můžeme přechodové i lexikální distribuce snadno aproximovat pomocí relativních četností v těchto datech. Emisní distribuci každého stavu tedy odhadneme pomocí relativní četnosti každého slova pozorovaného pro tento stav. Podobně přechodovou distribuci z každého stavu s pozorovaného v trénovacích datech aproximujeme pomocí relativní četnosti každého stavu s' pozorovaného jako následníka stavu s.

V případě, že v trénovacích datech nelze pozorovat jednoznačnou sekvenci stavů HMM, je nutné distribuce těchto skrytých stavů iterativně odhadnout. To by nastalo například tehdy, kdybychom v našem příkladě modelovali název produktu pomocí dvou stavů namísto jednoho. Motivací by zde mohla být snaha o to, aby jeden ze stavů lépe modeloval značku produktu (např. „Trek") a druhý název modelu (např. „Disc" nebo „04"). Stav pro „značku" by se zároveň měl naučit být před stavem pro „model". K tomu slouží tzv. *Baumův–Welchův algoritmus* neboli algoritmus *Forward–Backward*, což je varianta algoritmu *EM* pro učení bez učitele, např. (Manning, Schuetze, 2000). Tento nesupervizovaný algoritmus iterativně mění pravděpodobnostní distribuce modelu tak, aby v každém kroku zvýšil pravděpodobnost trénovacích dat v modelu HMM. Pro popis Baumova–Welchova algoritmu odkážeme například na práce (Manning, Schuetze, 2000), (Jelinek, 1997) a (Rabiner, 1989).

Pro praktické použití je navíc nutné nějak kompenzovat fakt, že každá trénovací data jsou pouze vzorkem dat, z kterých budeme trénovaný model skutečně extrahovat. V neviděných datech budou přechodové i emisní distribuce odlišné od trénovacích dat a navíc se v těchto datech budou vyskytovat nová slova, která nejsou zahrnuta v natrénovaných emisních distribucích (tj. jejich modelovaná pravděpodobnost by byla nesprávně 0). Proto se distribuce získané z relativních četností i pomocí nesupervizovaných algoritmů obvykle *vyhlazují*, tj. část pravděpodobnosti přiřazené jevům pozorovaným v trénovacích datech se přerozdělí ve prospěch jevů méně pozorovaných a jevů nepozoro-

vaných (např. každá emisní distribuce bude nově zahrnovat neznámé slovo). Přehled vyhlazovacích metod je popsán například v knize (Jelinek, 1997).

Vlastní extrakce informací z nového dokumentu spočívá v určení nejpravděpodobnější cesty, kterou by natrénovaný model musel projít, kdyby měl tento dokument generovat. Tento problém je efektivně řešen Viterbiho algoritmem (Manning, Schuetze, 2000), (Jelinek, 1997), (Rabiner, 1989), který pro dokument $D = w_1 ... w_n$ určí nejlepší posloupnost stavů $s_1 ... s_n$. Jelikož každý stav je asociován s nejvýše jednou sémantickou značkou, dostáváme zároveň i nejpravděpodobnější sémantické značkování dokumentu.

Princip Viterbiho algoritmu

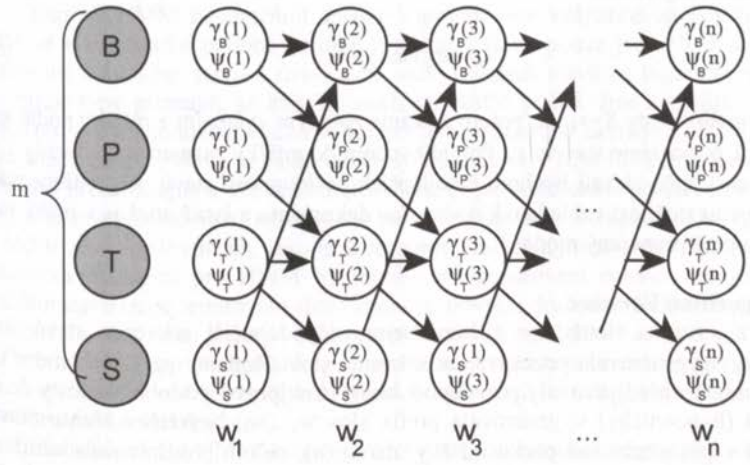
Mějme natrénovaný Markovův model μ a dokument $W = w_1 ... w_n$. Jde nám o určení nejpravděpodobnější posloupnosti stavů $S = s_1 ... s_n$ tímto modelem při (pomyslném) generování pozorovaného výstupu W:

$$S = \arg\max_{S'} P(S' \mid W, \mu) = \arg\max_{S'} \frac{P(S', W, \mu)}{P(W, \mu)} = \arg\max_{S'} P(S', W, \mu) , \tag{6.3}$$

kde

$$P(S' = s_1 \ldots s_n, W = w_1 \ldots w_n, \mu) = P(s_1 \mid s_{start}) P(w_1 \mid s_1) P(s_2 \mid s_1) P(w_2 \mid s_2) \ldots P(s_{n-1} \mid s_n) P(w_n \mid s_n) . \tag{6.4}$$

Triviální možností je *enumerovat všechny možné cesty* modelem μ o délce n, spočítat pro každou z nich pravděpodobnost a vybrat nejlepší cestu. Tento přístup však nelze použít, neboť již i pro náš jednoduchý model o pouhých čtyřech stavech je pro délku výstupu například 50 počet možných cest roven $4^{50} \cong 1{,}27.10^{30}$ (v případě plně propojeného modelu).



Obr. 6.16 Mřížka pro jednoduchý HMM sloužící k výpočtu nejlepší cesty.

Viterbiho algoritmus je aplikací dynamického programování, např. (Dasgupta, 2006), a využívá důsledku Markovovy vlastnosti HMM (Rabiner, 1989), a to konkrétně faktu, že při počítání pravděpodobnosti cesty $s_1...s_n$ lze využít dříve spočítanou pravděpodobnost kratší cesty $s_1...s_{n-1}$, která tvoří prefix počítané cesty.

Postup Viterbiho algoritmu lze ilustrovat pro náš jednoduchý HMM *mřížkou* (trellis) uvedenou na obr. 6.16. Mřížka znázorňuje všechny možné cesty modelem pro pozorovaný výstup a má tedy rozměry $n \times m$, kde n je délka výstupu (pomyslný čas t) a m je počet stavů modelu. Do každého bodu mřížky (t, s) budeme postupně ukládat pravděpodobnost nejlepší cesty, která v čase t vedla do stavu s. Tuto pravděpodobnost budeme značit $\gamma_s(t)$:

$$\gamma_s(t) = \max_{s_1,\ldots,s_{t-1}} P(s_1...s_{t-1}, s_t = s, w_1...w_t \mid \mu) . \tag{6.5}$$

Spolu s touto pravděpodobností ukládáme do bodů mřížky také zpětný index předchozího stavu této nejlepší cesty, který značíme $\psi_s(t)$. Hodnoty $\gamma_s(t)$ a $\psi_i(t)$ lze pro každý bod mřížky vypočítat indukcí počínaje časem 1:

$$\gamma_s(1) = P(s \mid s_{\text{start}}) P(w_1 \mid s) , \tag{6.6}$$

kde s_{start} je počáteční stav,

$$\gamma_s(t+1) = \max_{s'} \gamma_{s'}(t) P(s \mid s') P(w_{t+1} \mid s) \tag{6.7}$$

a zpětný index

$$\psi_s(t+1) = \arg\max_{s'} \gamma_{s'}(t) P(s \mid s') P(w_{t+1} \mid s) . \tag{6.8}$$

V momentě, kdy vypočteme hodnoty $\gamma_s(n)$ a $\psi_i(n)$ pro poslední sloupec mřížky, získáváme také poslední stav optimální cesty s_n, tedy ten stav, do kterého vede v čase n nejpravděpodobnější cesta

$$s_n = \arg\max_{s} \gamma_s(n) . \tag{6.9}$$

Zbytek optimální cesty $S=s_1...s_n$ potom získáme zpětným vyčtením z mřížky podle zpětných indexů ψ, počínaje stavem s_n. Pokud bychom do mřížky zaznamenávali vždy $k > 1$ nejlepších cest zpět, získali bychom k nejlepších posloupností stavů. Algoritmus má *lineární* časovou složitost vzhledem k *počtu slov* dokumentu a *kvadratickou* v *počtu stavů* modelu pro plně propojený model.

Princip algoritmu Forward

Viterbiho algoritmus sloužil ke zjištění nejpravděpodobnější sekvence stavů, která (hypoteticky) vygenerovala pozorovanou sekvenci slov. Hodnoty $\gamma_s(t)$ ukládané v každém bodě mřížky představovaly pravděpodobnost té nejpravděpodobnější cesty do stavu s, která (hypoteticky) vygenerovala prefix slov $w_1 \ldots w_t$. Největší z těchto pravděpodobností v posledním sloupečku mřížky, $\max_s \gamma_s(n)$, potom představovala sdruženou pravděpodobnost $P(W, \textit{nejlepší } S \textit{ pro } W)$, tedy pravděpodobnost, se kterou model hypoteticky vygeneroval celou pozorovanou sekvenci slov W za předpokladu, že přitom

procházel nejlepší (nejpravděpodobnější) stavovou cestou, jež tento výstup mohla vygenerovat. Pokud v rovnici (6.7) nahradíme výběr maxima sumou, dostaneme místo hodnot $\gamma_s(t)$ tzv. *pravděpodobnost Forward* $\alpha_s(t)$:

$$\alpha_s(t+1)=\sum_{s'}\alpha_{s'}(t)P(s \mid s')P(w_{t+1} \mid s)\,. \tag{6.10}$$

Pravděpodobnost Forward $\alpha_s(t)$ zde reprezentuje pravděpodobnost vygenerování prefixu slov $w_1 \ldots w_t$ a skončení ve stavu s bez ohledu na to, kterou stavovou cestou model během generování procházel (pravděpodobnosti veškerých možných stavových cest vedoucích do stavu s jsou sečteny). Pokud dále sečteme všechny pravděpodobnosti Forward v jednom sloupečku mřížky (v čase t), dostáváme celkovou pravděpodobnost vygenerování slov $w_1 \ldots w_t$ naším modelem bez ohledu na zúčastněné stavy. Suma v posledním sloupečku mřížky pak obsahuje pravděpodobnost celé vstupní sekvence W v modelu.

Aplikací algoritmu Forward je několik. Nejvýznamnějším je jeho použití pro trénink v rámci zmíněného algoritmu Forward–Backward pro trénování HMM. Algoritmus Forward lze také použít pro klasifikaci textu. Máme-li například několik HMM natrénovaných na různých typech textu, můžeme srovnávat pravděpodobnosti (hypotetického) vygenerování nového klasifikovaného textu každým z těchto modelů. Text je potom klasifikován jako příslušný k tomu HMM, který ho s největší pravděpodobností vygeneroval. Konkrétní použití může být například pro jednoduchou klasifikaci textu do několika hrubých významových skupin (např. při automatické klasifikaci příchozích hovorů v call centrech) nebo při klasifikaci jazyka textu, kde místo slov HMM generují jednotlivé znaky.

Omezení HMM jako extrakčních modelů

Skryté Markovovy modely trpí několika vážnými nedostatky. Jako nejvýraznější uvádějí autoři práce (McCallum a kol., 2000) tyto dva:

Zaprvé, HMM neumožňují snadné zapojení více vzájemně závislých příznaků. HMM ve své základní podobě emituje v každém stavu pouze jednotlivé slovo. Slovo generované v určitém stavu a dvojice po sobě jdoucích stavů (u bigramového HMM) jsou jediné typy příznaků, se kterými základní HMM počítá. Jiné algoritmy IE přitom dokáží pro své rozhodování využít velké množství vzájemně závislých příznaků. Kromě právě značkovaného slova se často využívají také slova v jeho blízkém i dalekém okolí (např. dvě předcházející a dvě následující slova, popř. interpunkci použitou pro ukončení věty), vlastnosti těchto slov (např. různé morfologické nebo syntaktické kategorie, členství těchto slov ve slovnících, jako jsou seznamy křestních jmen) nebo formátování částí dokumentu. Řešením pro HMM by mohlo být generování celého vektoru příznaků v každém stavu (např. emitování slova spolu se slovním druhem a členstvím ve známých slovnících), avšak tento přístup vede k rychlému prořídnutí trénovacích dat. Tomu by se dalo bránit zavedením předpokladu podmíněné nezávislosti příznaků, avšak ta často není v praxi splněna (vezměme například příznaky „slovo začíná velkým písmenem" a „slovo je na seznamu křestních jmen").

Zadruhé, i když HMM dávají navzdory své jednoduchosti často překvapivě dobré výsledky (tj. nejpravděpodobnější cesta nalezená Viterbiho algoritmem bývá správná), často se u HMM vyskytují problémy s interpretací pravděpodobnosti nalezené cesty.

Viterbiho algoritmus hledá sekvenci stavů S maximalizující pravděpodobnost $P(S, W)$ pro dané W, a proto pravděpodobnost nejlepší cesty S ve Viterbiho algoritmu zahrnuje jak pravděpodobnost značkování, tak pravděpodobnost vygenerování pozorované sekvence W (ze všech možných jiných sekvencí slov). Generativní a tudíž i celková pravděpodobnost výsledné cesty je tedy velmi malá, což ztěžuje její interpretaci. Je nicméně možné pravděpodobnosti alternativních cest mezi sebou porovnávat a usuzovat na to, kolikrát je v daném HMM jedno značkování S stejného vstupu W pravděpodobnější než jiné značkování S'. Pokud chceme v HMM navíc získat podmíněnou pravděpodobnost $P(S|W)$ určité cesty, je nutné zjistit pomocí popsaného algoritmu Forward ještě hodnotu $P(W)$. Potom vypočteme $P(S|W) = P(S, W)/P(W)$. Kvůli zmíněné nutnosti ignorovat relevantní příznaky však HMM často nemodeluje tyto pravděpodobnosti dobře, a tak i když je nejlepší cesta správná, nemusí její pravděpodobnost, resp. poměr její pravděpodobnosti k pravděpodobnostem soupeřících cest, odpovídat realitě.

6.5.3.2 Markovovy modely s maximální entropií

Markovovy modely s maximální entropií (Maximum Entropy Markov Models, MEMM), navržené v práci (McCallum a kol., 2000) jako řešení nedostatků HMM, se od HMM liší tím, že nahrazují *generativní lexikální distribuci* $P(w|s)$ pro každý stav s distribucí opačnou, *podmíněnou*: $P(s|F(w))$, jak je patrné z obr. 6.14, části b). Zde je w slovo, jehož sémantickou třídu předpovídáme, a $F(w)$ je množina příznaků naplněná hodnotami pro toto slovo. Příznaky pro slovo w mohou být přitom definovány na celé vstupní větě W. V praxi se využívají binární příznaky a součástí definice příznaku je i předpovídaná třída, například:

– současné slovo = „Trek" a předpovídaná třída je „název produktu" nebo
– předchozí dvě slova = „zlevněno na" a předpovídaná třída je „cena".

Generativní HMM modelovaly sdruženou pravděpodobnostní distribuci $P(W, S)$ popisující pravděpodobnost výskytu textového řetězce W s určitým označkováním S. MEMM se vyhýbají modelování pravděpodobnosti samotného textového řetězce W a modelují přímo podmíněnou pravděpodobnost $P(S|W)$, která nás z hlediska předpovědi S nejvíce zajímá. Při extrakci pomocí MEMM jde tedy (stejně jako u HMM) o zjištění nejpravděpodobnější sekvence stavů pro pozorovaná slova W při použití natrénovaného modelu μ (viz rovnice 6.11). Distribuce $P(S|W)$ je však modelována jako součin pravděpodobností jednotlivých stavů podmíněných předchozím stavem a příznaky definovanými na celém vstupu:

$$\begin{aligned} &P(S' = s_1 \ldots s_n \,|\, W = w_1 \ldots w_n, \mu) \\ &= P(s_1 \,|\, s_{start}, F(w_1)) P(s_2 \,|\, s_1, F(w_2)) \ldots P(s_n \,|\, s_{n-1}, F(w_n)). \end{aligned} \tag{6.11}$$

Zde $F(w_k)$ představuje množinu hodnot příznaků $f(W, k, s_{k-1}, s_k)$ pro aktuální slovo w_k a $P(s_k|s_{k-1}, F(w_k))$ značí pravděpodobnost značky s_k slova w_k při daném předchozím stavu s_{k-1} a při dané pozorované množině příznaků pro aktuální slovo w_k. Pravděpodobnost přechodu ze stavu s_{k-1} do stavu s_k je v MEMM modelována jako normalizovaný exponenciální součet těch vah binárních příznaků pojících se se zdrojovým stavem s_{k-1} a s cílovým stavem s_k, které byly na pozici slova w_k pozorovány. O součet vah jde proto,

že příznaky jsou binární, přičemž platí, že $f(W, k, s_{k-1}, s_k) = 1$, popř. 0, pro pozorovaný nebo chybějící příznak

$$P(s_k \mid s_{k-1}, F(w_k)) = \frac{1}{Z(s_{k-1}, w_k)} \exp \sum_{f \in F(w_k)} \lambda_{f, s_k, s_{k-1}} f(W, k, s_{k-1}, s_k) . \quad (6.12)$$

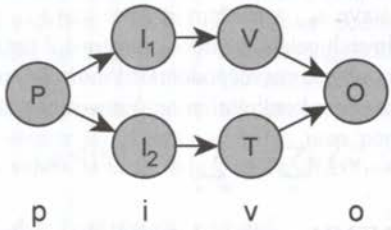
Zde hodnota λ reprezentuje natrénovanou váhu konkrétního příznaku f asociovaného s hodnotou předchozího stavu s_{k-1} a předpovídaného stavu s_k. Připomeňme, že příznak může být definován na slovech celého vstupu. Proměnná Z hraje pouze roli normalizačního faktoru, zajišťujícího, aby se pravděpodobnosti možných předpovídaných stavů pro konkrétní pozorované příznaky a konkrétní minulý stav sčítaly do 1:

$$Z(s_{k-1}, w_k) = \sum_{s_k} \exp \sum_{f \in F(w_k)} \lambda_{f, s_k, s_{k-1}} f(W, k, s_{k-1}, s_k) . \quad (6.13)$$

Aplikace natrénovaného MEMM je relativně jednoduchá – Viterbiho algoritmus probíhá pro MEMM zcela analogicky jako pro HMM. Složitější je však trénování vah příznaků. To se provádí metodou maximální věrohodnosti tak, aby natrénovaný model přiřazoval sekvencím značek z trénovacích dat co největší pravděpodobnost při určitých omezeních. Tato omezení spočívají v tom, že pravděpodobnost výskytu každého příznaku v modelu má být stejná jako jeho empirická relativní četnost pozorovaná v trénovacích datech. Optimalizace za těchto podmínek vede k naučení distribuce, která je v souladu s pozorovanými četnostmi použitých příznaků, jinak je ale co nejvíce neurčitá – má maximální entropii – odtud také pochází jméno celého modelu. Při tréninku se tedy maximalizuje hodnota rovnice (6.10) přes celá trénovací data. Tento optimalizační problém je nutné řešit iterativními metodami, např. *Generalized Iterative Scaling* (McCallum a kol., 2000), nebo gradientními metodami, z nichž se nejčastěji využívá *L-BFGS* (Byrd a kol., 1995).

Ačkoliv MEMM umožňují práci s vysokým počtem navzájem závislých příznaků, mají oproti generativním HMM také nevýhody. Jako hlavní nevýhodu uvádí Lafferty (Lafferty, 2001) tzv. *label bias*, tedy systematickou preferenci některých značek před jinými. Jde o to, že pravděpodobnost značkování $P(S' \mid W, \mu)$ je modelována jako součin lokálních podmíněných pravděpodobností typu $P(s_k \mid s_{k-1}, F(w_k))$. Problém je právě v tom, že lokální distribuce jsou podmíněny pozorovaným slovem a příznaky definovanými vzhledem k němu a nevyjadřují se vůbec o pravděpodobnosti samotných pozorovaných příznaků v kontextu předchozího stavu s_{k-1}. Tedy lokální pravděpodobnostní distribuce nedokáží zachytit, že určité pozorované příznaky „nepasují" na předchozí stav, neboť mají pozorované příznaky v podmiňující části. Toto je markantní zejména u stavů s několika málo možnými následnými stavy. Extrémním případem jsou stavy s jediným možným přechodem na další stavy. Takové stavy s_{k-1} efektivně vedou k ignorování pozorovaných příznaků pro (jediný) následný stav s_k, protože lokální pravděpodobnostní distribuce zde musí nutně být $P(s_k \mid s_{k-1}, F(w_k)) = 1$, nehledě na hodnoty pozorovaných příznaků pro w_k, které mohou být v rozporu se stavem s_k. V praxi to znamená, že pokud se Viterbiho algoritmus na začátku „vydá" po špatné cestě, další stavy na takové cestě

(obzvlášť mají-li málo následných stavů) nejsou schopny odmítnout jim nepříslušná pozorování (tj. přiřadit jim nízkou pravděpodobnost, a tím snížit pravděpodobnost takové špatné cesty). Obecně tedy platí, že MEMM zvýhodňují stavy s méně možnými následnými přechody před stavy s více následnými přechody, přesněji řečeno jsou preferovány stavy s nižší entropií pravděpodobnostního rozdělení následných stavů.



Obr. 6.17 Problém „Label bias".

Předpokládejme příklad jednoduchého modelu MEMM o 6 stavech znázorněného na obr. 6.17, který přiřazuje pozorovaným sekvencím malých písmen sémantické značky sestávající vždy z odpovídajících velkých písmen. Tento model natrénujme trénovacími daty sestávajícími z 10 příkladů slova „pivo", které mají procházet odpovídající „horní" cestou modelu se značkováním „PI_1VO", a z 10 příkladů slova „pito", které mají procházet odpovídající spodní větví se značkováním „PI_2TO". Příznakem budiž pouze identita aktuálního symbolu. Snadno nahlédneme, že na výstupu ze stavu P dojde k natrénování distribuce $P(I_1|P, F(pivo)) = P(I_2|P, F(pivo)) = 0{,}5$ a stejně tak pro případ řetězce „pito". Dvě distribuce vedoucí ze stavů I_1 a I_2 (a podobně ze stavů V a O) nemají jinou možnost než bez ohledu na pozorovaný příznak přejít na jediný možný další stav s pravděpodobností 1. Uvedený model tak oba řetězce nedokáže rozlišit. Pravděpodobnost obou značkování bude pro oba řetězce 50%. Pokud bychom zde použili generativní HMM se stejnou strukturou, generativní emisní distribuce stavů V a T by průchodu nesprávnou cestou zabránily.

6.5.3.3 Podmíněná náhodná pole – Conditional Random Fields

Východiska z problému „label bias" popsaného v předchzím výkladu jsou dle Laffertyho (Lafferty, 2001) dvě: buď neomezovat strukturu přechodů modelu a pracovat s plně propojenými modely, nebo převést lokální pravděpodobnostní distribuce $P(s_k|s_{k-1}, F(w_k))$ na nějaké jiné (nepravděpodobnostní) funkce vracející *váhy*, které nemusejí přes následující stavy sčítat do 1. Druhé řešení představují *Conditional Random Fields* (*CRF*).

CRF jsou neorientovaným grafickým modelem, který odstraňuje popsaný problém „label bias" a přitom stále umožňuje využít velké množství vzájemně závislých příznaků podobně jako MEMM. CRF jsou speciálním případem neorientovaného grafického modelu, kde náhodné veličiny reprezentující značku na určité pozici dokumentu jsou lineárně propojeny a každá z nich je dále propojena s aktuálními pozorováními; viz obr. 6.14, část c).

Stejně jako MEMM modelují i CRF přímo podmíněnou pravděpodobnostní distribuci, v rámci níž se snažíme nalézt nejpravděpodobnější označkování vstupní sekvence W:

$$S = \arg\max_{S'} P(S' \mid W, \mu) . \tag{6.14}$$

Na rozdíl od MEMM však CRF tuto pravděpodobnost nemodelují součinem jednotlivých lokálních podmíněných distribucí předpovídajících další stav na základě aktuálních pozorování hodnot příznaků a předchozího stavu. Lokální podmíněné distribuce $P(s_k|s_{k-1}, F(w_k))$ jsou zde nahrazeny tzv. *potenciálovými funkcemi*, které samy nejsou normalizované (chybí zde normalizační faktor Z). Normalizace se v CRF provádí až pro výslednou podmíněnou pravděpodobnostní distribuci značkování celé pozorované sekvence slov W – tím se brání lokálnímu problému „label bias". Výhodou CRF oproti MEMM tedy je, že dokáží (podobně jako generativní modely) za pozorování nevyhovující uvažovanému stavu penalizovat i takové cesty modelem, ve kterých je počet přechodů na další stavy nízký. Pravděpodobnost označkování sekvence W je konkrétně modelována jako normalizovaný exponenciální součet vah pozorovaných příznaků, definovaných přes celou sekvenci W, vždy vzhledem k aktuálnímu slovu. Příznakové funkce jsou zde, stejně jako v případě MEMM, bigramové, protože berou v úvahu vždy předpovídanou a předchozí značku

$$P(S' = s_1 \ldots s_n \mid W = w_1 \ldots w_n, \mu) = \frac{1}{Z(\lambda, W)} \exp \sum_{i=1}^{n} \sum_{f \in F} \lambda_{f, s_i, s_{i-1}} f(W, i, s_{i-1}, s_i) , \tag{6.15}$$

kde Z je normalizační faktor

$$Z(\lambda, W) = \sum_{S} \exp \sum_{i=1}^{n} \sum_{f \in F} \lambda_{f, s_i, s_{i-1}} f(W, i, s_{i-1}, s_i) . \tag{6.16}$$

Stejně jako u MEMM představují hodnoty λ natrénované váhy konkrétního příznaku f. Neorientovaný grafický model pro CRF je znázorněn na obr. 6.14, v části c), která ukazuje výřez ze sekvence slov a stavů. Kliky v tomto grafu jsou dvojího druhu: vždy dva sousední stavy, dále stav a jeho slovo. Pokud bychom potřebovali příznakové funkce vztahující se zároveň ke slovům a k předchozímu a současnému stavu, doplnili bychom do grafického modelu ještě spojnici mezi minulým stavem S_{t-1} a $F(W)$, čímž spojíme oba typy klik do jednoho „trojúhelníkového" typu klik, který je uvažován v rovnicích (6.15) a (6.16). Potenciálové funkce Ψ grafického modelu jsou pro každou trojúhelníkovou kliku ve tvaru

$$\psi(s_{i-1}, s_i, F(W)) = \exp \sum_{f \in F} \lambda_{f, s_i, s_{i-1}} f(W, i, s_{i-1}, s_i) . \tag{6.17}$$

Každý neorientovaný grafický model definuje svou pravděpodobnostní distribuci jako normalizovaný součin potenciálových funkcí všech svých klik. Také pro CRF o délce n slov platí, že vynásobením n potenciálových funkcí trojúhelníkových klik z rovnice

(6.17) dostaneme výraz, který po normalizaci odpovídá rovnici (6.15). Násobení exponenciálních potenciálových funkcí zde převedeme na součet vah pozorovaných příznaků uvnitř exponentu

$$P(S' = s_1 \ldots s_n \mid W = w_1 \ldots w_n, \mu) = \frac{1}{Z(\lambda, W)} \prod_{i=1}^{n} \psi(s_{i-1}, s_i, F(W))$$

$$= \frac{1}{Z(\lambda, W)} \exp \sum_{i=1}^{n} \sum_{f \in F} \lambda_{f, s_i, s_{i-1}} f(W, i, s_{i-1}, s_i) . \quad (6.18)$$

Pro určení nejpravděpodobnější sekvence značek pomocí natrénovaného CRF stejně jako v případě MEMM použijeme Viterbiho algoritmus. Trénink vah se provádí podobně jako u MEMM metodou maximální věrohodnosti, tj. maximalizací přímo modelované podmíněné pravděpodobnosti sekvencí značek v celých trénovacích datech, rov. (6.15). Zatímco v případě MEMM lze zvlášť optimalizovat jednotlivé přechodové distribuce mezi každou dvojicí stavů $P(s_k|s_{k-1}, F(w_k))$, je v případě CRF optimalizační úloha těžší o to, že trénujeme pouze jednu globální distribuci s velmi vysokým počtem parametrů. Pro optimalizaci se zde používají gradientní algoritmy, jako je *L-BFGS* (Byrd a kol., 1995).

6.5.3.4 Jiné grafické modely

Z dalších grafických modelů, o kterých se v této kapitole nezmíníme, se jeví jako velmi perspektivní pro IE tzv. *Markov Logic Networks* (Richardson, Domingos, 2006), který kombinuje grafické modely s logikou prvního řádu. Velmi dobrých výsledků dosahují také *bayesovské sítě*, které spadají mezi orientované grafické modely (Peshkin, Pfeffer, 2003). Mimo grafické modely je v oblasti IE populárním klasifikátorem také *SVM*, viz např. (Takeuchi, Collier, 2002).

6.6 Využití extrakčních metod pro sémantizaci webu

S ohledem na tematiku předcházejících kapitol je vhodné se zamyslet nad perspektivami aplikace extrakce informací při převodu informací dostupných na běžném „textovém" webu do formátů *sémantického webu*.

Jedním z rysů aplikace IE v prostředí sémantického webu je dostupnost *ontologických modelů*, které mohou jednak (zejména svou lexikální složkou) napomoci při extrakci samotné, jednak sloužit jako abstraktní schéma pro struktury, do nichž jsou vyextrahovaná data ukládána. Využití ontologií v IE jsme si přiblížili v části 6.4.1.1 a pro specifickou technologii extrakčních ontologií v odst. 6.5.1, nebudeme se mu proto dále podrobněji věnovat. Přehledová studie (Uren a kol., 2006), která vyšla z analýzy širokého spektra existujících extrakčních projektů a systémů, zařazuje do množiny požadavků kladených na extrakční systémy pro znalostní management v prostředí *sémantického webu* vedle podpory ontologií a *standardních datových formátů* (zejména RDF) dále

kolaborativnost, návaznost na běžné *pracovní prostředí* uživatele, pokrytí *heterogenních* dokumentových formátů a také podchycení dokumentů s postupně se *vyvíjejícím* obsahem. Většina z těchto požadavků se však vztahuje spíše k nadstavbovým aplikacím než k vlastní extrakci z textu.

Zdá se, že v principu lze téměř každou z heterogenních úloh IE, popsaných v tab. 6.1 (lišících se množstvím a homogenitou vstupního formátování, gramatickou správností textu, nejednoznačností či jednoznačností extrahovaných hodnot apod.), považovat za relevantní pro realizaci sémantického webu. Ne ve všech případech lze ovšem u metod IE předpokládat dostatečnou efektivitu a spolehlivost, zejména v porovnání s alternativními postupy – na jedné straně s ručním anotováním, na straně druhé s přímým převodem existujících strukturovaných dat (zejména z relačních databází) do formátů sémantického webu bez „textového" mezikroku. V praxi lze navíc očekávat, že se automatická IE bude s ručním zpracováním i s automatickým převodem ze „skrytých" databází doplňovat. Proto zřejmě v blízké budoucnosti zřetelně vykrystalizuje jen několik scénářů využití IE, které již jsou, nebo v dohledné budoucnosti budou ve významném rozsahu nasaditelné do praxe.

Prvním scénářem je automatický sběr *jednoduchých údajů generického typu* z velkého objemu webového (či jiného) textu. Tato úloha je rutinně řešena například roboty sbírajícími webové a e-mailové odkazy, na vyšší úrovni pak nástroji pro *rozpoznávání pojmenovaných entit* (zejména jmen osob, organizací, míst apod.), z nich je v současnosti populární zejména *OpenCalais*[17] firmy Thomson Reuters a nástroj slovinské firmy *Zemanta*[18], obdobným nástrojem je také *Epiphany*[19] z německého centra DFKI. Všechny tři nástroje generují výsledky ve formátu RDF a předpokládá se jejich začlenění do datových zdrojů Linked data (viz odst. 2.4.6). Tvorbu složitějších datových struktur tím částečně přesouvají z vlastní analýzy textu[20] do následného zpracování již v rámci znalostních technik sémantického webu.

Dalším scénářem, který se masově uplatňuje již dnes (byť často primitivní formou jednorázově vyvíjených procedurálních skriptů spíše než aplikací specializovaných nástrojů IE), je nasazení *wrapperových technik* (viz odst. 6.4.3) na data s vysokou regularitou značkování HTML, jako jsou přehledové tabulky elektronických obchodů. Jak už bylo řečeno, tento scénář chápe extrakci z každého webového sídla jako samostatnou úlohu se značnou režií, a proto se vyplatí jen pro sídla rozsáhlejší nebo taková, kde je vysoká frekvence obměňování obsahu. Přesnost wrapperového přístupu je natolik vysoká, že ho lze využít pro komerčně významné úlohy, jako je komparativní nakupování. Na druhou stranu ovšem postihuje jen omezený okruh informací, které jsou zpřístupňovány ve „tvrdě strukturované" podobě. Je také nutné říci, že potenciál tohoto přístupu pro standardní postupy sémantizace webu je nižší kvůli rychlému zastarávání dat, která se tudíž často nevyplatí mapovat na podrobné ontologické modely a shromažďovat do distribuovaných úložišť. Spíše se proto uplatňují centralizované portály.

Odlišným scénářem, který má s předchozím postupem společnou orientaci na rozsáhlejší bloky údajů v HTML, je automatická *identifikace informací*, které budou

[17] http://www.opencalais.com
[18] http://zemanta.com
[19] http://projects.dfki.uni-kl.de/epiphany
[20] V zájmu pravdy uveďme, že OpenCalais se z textu pokouší extrahovat i relace. Protože však nedisponuje specifickými znalostmi o doméně a korpusu dokumentů, relace jsou zpravidla vymezeny vágně a opírají se o ortografické („povrchové") vlastnosti textu (např. přítomnost určitého slovesa ve větné konstrukci).

následně (přímo v kontextu daného webového sídla) zpracovány *ručně*. Tento scénář se uplatní zejména při expertním hodnocení webových sídel s ohledem na jejich kvalitu, důvěryhodnost apod. Nedávno skončený projekt EU *MedIEQ*[21], jehož technologickým středem byla právě extrakce informací, usiloval o podporu specializovaných lékařských agentur, které přidělují akreditační značky zdravotnickým webovým stránkám (Karkaletsis a kol., 2008). Extrakční modely byly (ať už ručně nebo automaticky) vyvinuty pro rozpoznávání kritérií významných pro udělování těchto značek, například přítomnosti kontaktní informace na zodpovědného lékaře či správce webu, formuláře pro virtuální konzultaci nebo informace o způsobu nakládání s důvěrnými daty. Systém nenahrazuje lidského uživatele – hodnoticího experta, ale vykoná za něj mechanickou práci spočívající v procházení webového sídla a vyhledávání příslušných informací a také předvyplní údaje do „protokolu o hodnocení", který je součástí strojově čitelných informací akreditační značky. Tyto informace jsou přístupné na sémantickém webu díky vazbě na ontologický slovník W3C *POWDER*[22].

Posledním scénářem, o němž se zmíníme, je využití IE ke *zjemňování granularity dat*, která jsou již strukturovaná. Prvotní struktura přitom může pocházet přímo z dat sémantického webu (zdroje Linked Data, jako je například dataset *Linked Open Commerce*[23]) nebo z předcházející sémantické analýzy hierarchické struktury (stromu DOM) značek HTML na webové stránce, např. pomocí wrapperové extrakce. Na rozdíl od extrakce z rozsáhlejších dokumentů se zde analýza může opřít o kompaktní sémantiku celého analyzovaného textu, kterým je v případě webových prezentací firem například popis výrobku, služby nebo činnosti firmy, u informací o lidech můžeme jako příklad údaje s hrubou granularitou uvést celé jméno člověka s tituly a pracovní pozicí. Relevantní ontologie a s nimi spojené metodiky, jako je *GoodRelations* (Hepp, 2008), (viz též odst. 2.4.5), explicitně připouštějí různou míru granularity údajů stejného typu, kterou lze postupně zpřesňovat ručními nebo automatickými metodami.

6.7 Závěrečné poznámky

Cílem této kapitoly bylo uvést čtenáře do problematiky sémantického anotování a poskytnout přehled současných metod pro extrakci informací. Vzhledem k rostoucímu objemu textových informací na internetu, a to i v rámci firemních intranetů, je o aplikace využívající sémantické anotace poměrně velký zájem. Primárním cílem bývá zefektivnit vyhledávání v pokrytých dokumentech. Jako řadu dalších aplikací umělé inteligence lze však i extrakci informací zneužít například pro podporu cenzury nebo pro nežádoucí dolování dat z webových sídel.

Z konkrétních extrakčních metod zažívají v současnosti velký rozvoj zejména aplikace grafických pravděpodobnostních modelů a přístupy využívající bootstrapping pro nesupervizované učení. Ve webovém prostředí je významná vazba na ontologické modely a také provázání s již strukturovanými daty, zejména ve formátu Linked Data.

Pro českého čtenáře by mohla být užitečnou informace o některých tuzemských projektech, která se týká extrakce informací a sémantické anotace. Problém extrakce po-

[21] http://www.medieq.org
[22] http://www.w3.org/2007/powder
[23] http://linkedopencommerce.com/

jmenovaných entit v češtině řeší publikace (Kravalová, Žabokrtský, 2009) a (Ševčíková a kol., 2007). Autoři prezentují návrh systematického členění obecných druhů pojmenovaných entit do dvojúrovňové hierarchie, kde na detailní úrovni je předmětem extrakce více než 60 druhů pojmenovaných entit (např. název městské části, název TV stanice, směrovací číslo). Experimenty proběhly nad datovou kolekcí 6000 ručně anotovaných vět vybraných z Českého národního korpusu[24], obsahujících přibližně 33 000 výskytů pojmenovaných entit. Jako extrakční algoritmus byl použit klasifikátor SVM, který byl aplikován na okna (sousloví) v textu různých délek. Podle našeho členění z odst. 6.4.2 tedy jde o subsymbolický klasifikátor izolovaných příkladů aplikovaný na vstupní věty reprezentované jako svazy sousloví různých délek. Pro každou délku sousloví byl natrénován zvláštní klasifikátor SVM. Jako příznaky (extrakční indicie) byly využity různé morfologické a ortografické kategorie extrahovaných slov a slov v jejich okolí, členství těchto slov v seznamech známých hodnot, využita byla také lematizace. Klasifikace dosáhla F-míry kolem 70 %, což je relativně dobrý výsledek vzhledem k vysokému počtu vzájemně si blízkých druhů pojmenovaných entit a (pro některé druhy entit) omezenému množství dat.

Hajič (Hajič a kol., 2004) dále popisuje proces manuálního sémantického anotování českého textu z Pražského závislostního korpusu pomocí významů definovaných v české verzi WordNetu (Smrž, 2004). Všechna plnovýznamová slova ve vstupním textu (podstatná, přídavná jména a slovesa), vyskytující se v českém WordNetu, byla člověkem anotována právě jedním významem ze všech možných významů daného slova ve WordNetu (anotace pomocí tzv. synsetů). Takto bylo anotováno přibližně 11 000 vět. Ze všech nalezených plnovýznamových slov byla ve WordNetu zachycena přibližně polovina a z těchto slov bylo 70 % potenciálně víceznačných (majících ve WordNetu více než 1 synset). Zajímavé pozorování je, že – pravděpodobně díky vysoké granularitě a přesahu významů mezi synsety WordNetu – anotátorská shoda dosáhla mezi dvěma anotátory jen 70 %, tedy zhruba třetina slov nebyla anotována spolehlivě.

Pro manuální anotování doménově specifických textů (zdravotnických záznamů) byl na FEL ČVUT vyvinut nástroj *MedAT* [25]. Nástroj využívá vazbu na lékařskou ontologii, podle které jsou vyextrahovaná data ukládána, a také mohou být pro další zpracování převedena do relační databáze.

Extrakcí informací pojmenovaných entit v širším smyslu z gramatického textu se rovněž zabývají práce (Dědek, 2010), (Dědek a Vojtáš, 2009). Autoři aplikovali induktivní logické programování (ILP) pro indukci extrakčních pravidel nad závislostními tektogramatickými větnými stromy. Pro konstrukci těchto stromů byl využit závislostní větný parser trénovaný pomocí Pražského závislostního korpusu (Hajič, 2004), integrovaný společně s dalšími lingvistickými komponentami v systému TectoMT (Žabokrtský a kol., 2008); tento systém byl vyvinut jako platforma pro analýzu českých i cizojazyčných textů, mimo jiné za účelem strojového překladu. Zajímavá je zde reprezentace dokumentu, kde klasifikovanými příklady jsou přímo uzly tektogramatických stromů, přičemž klasifikace uzlu jako relevantní pojmenované entity znamená extrakci řetězce slov náležejících pod tento uzel. Autoři prezentují předběžné výsledky metody na sadě 50 dokumentů popisujících nehody; extrahovanou entitou je například popis vzniklé škody.

[24] http://ucnk.ff.cuni.cz
[25] http://nit.felk.cvut.cz/projects/medical-annotation-tool-medat

Vizuální analýzou stránek HTML za účelem extrakce typických významových oblastí se zabývá studie (Kunc a Burget, 2008). Autoři využívají metodu hierarchického shlukování vizuálních oblastí v renderovaném dokumentu HTML a klasifikují tyto oblasti pomocí různých klasifikátorů do doménově závislých kategorií. V prezentované studii analyzující novinové stránky jde o identifikaci titulu stránky, nadpisů článků, textů aktualit, navigačních menu apod.

Autoři tohoto textu experimentovali, v rámci široce zaměřeného projektu analýzy webových dat, nazvaného *Rainbow*[26], s využitím skrytých Markovových modelů (HMM) pro extrakci atributů nabídek produktů na internetu. Výsledky pro doménu prodeje jízdních kol shrnuje studie (Labský a kol., 2005) s aplikací pro strukturované vyhledávání. Kromě názvu prodávaného modelu a ceny bylo předmětem extrakce zhruba deset dalších specifických atributů včetně případného obrázku produktu. Pro extrakci obrázků byly do procesu značkování dokumentu zapojeny také trénované klasifikátory obrázků založené na velikosti, tvaru a obsahu obrázku. Experimenty proběhly nad kolekcí přibližně 100 heterogenních webových stránek internetových obchodů, obsahující asi 1000 nabídek sestávajících z 4100 výskytů pojmenovaných entit a 1500 výskytů obrázků.

Zkušenosti z projektu *Rainbow* byly následně využity při vývoji metody extrakčních ontologií, popsané v odst. 6.5.1, a také v projektu EU MedIEQ, zmíněného v oddílu 6.6, kde se uplatnily jak extrakční ontologie, tak grafické modely typu CRF.

Literatura

Agichtein E., Gravano A.: Snowball: Extracting relations from large plain-text collections. In: Proc. *5th ACM Intl. Conf. on Digital Libraries*, 2000, s. 85–94.

Baumgartner R., Wien D. T., Flesca S., Gottlob, G.: Supervised Wrapper Generation with Lixto. In: *Proc. VLDB Demo*, 2001.

Berka P.: *Dobývání znalostí z databází*. Academia, Praha 2003.

Berners-Lee T., Hendler J., Lassila O.: The Semantic Web. In: *Scientific American Magazine*, 2001.

Bisani M., Vozila P., Divay O., Adams J.: Automatic Editing in a Back-End Speech-to-Text System. In: *Proc. ACL-08: HLT*, 2008, s. 114–120.

Bishop C. M.: *Pattern Recognition and Machine Learning*. Springer, 2006.

Borkar V., Deshmukh K., Sarawagi S.: Automatic segmentation of text into structured records. In: *Proc. SIGMOD*, 2001.

Brill E.: Some Advances in Transformation-Based Part of Speech Tagging. In: *Proc. The 12th National Conference on Artificial Intelligence*, 1994.

Brin S.: Extracting patterns and relations from the World-Wide Web. In: *Proc. Intl. Workshop on Web and Databases*, 1998.

Byrd R. H., Lu P., Nocedal J.: A Limited Memory Algorithm for Bound Constrained Optimization. In: *SIAM Journal on Scientific and Statistical Computing*, 16, 5, 1995, s. 1190–1208.

[26] http://rainbow.vse.cz

Califf M. E., Mooney R. J.: Relational Learning of Pattern-Match Rules for Information Extraction. In: *Proc. Sixteenth National Conference on Artificial Intelligence* (AAAI-99), 1999, s. 328–334.

Cimiano P., Staab S.: Learning by Googling. In: *Proc. SIGKDD Explorations*, 2004.

Ciravegna F.: LP2 – an adaptive algorithm for information extraction from web-related texts. In: *Proc. IJCAI*, 2001.

Ciravegna F., Chapman S., Dingli A., Wilks Y.: Learning to Harvest Information for the Semantic Web. In: *Proc. ESWS*, 2004.

Ciravegna F., Dingli A., Guthrie D., Wilks Y.: Integrating Information to Bootstrap Information Extraction from Web Sites. In: *Proc. IJCAI Workshop on Information Integration on the Web*, 2003.

Cohen W.: *Information Extraction and Integration*: an Overview, 2004. Online tutoriál, http://www.cs.cmu.edu/~wcohen/ie-survey.ppt (17. 11. 2010).

Crescenzi V., Mecca G., Merialdo P.: RoadRunner: towards automatic data extraction from large Web sites. In: *Proc. 26th International Conference on Very Large Database Systems (VLDB)*, 2001, s. 109–118.

Cunningham H., Maynard D., Bontcheva K., Tablan V.: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In: *Proc. The 40th Anniversary Meeting of the Association for Computational Linguistics (ACL)*. Philadelphia, 2002.

Dasgupta S., Papadimitriou C., Vazirani U.: *Algorithms*. McGraw-Hill, 2006.

Dědek J.: Towards semantic annotation supported by dependency linguistics and ILP. In: *Proc. 9th Intl. Semantic Web Conference*, LNCS, vol. 6497, 2010, s. 297–304.

Dědek J., Vojtáš P.: Web information extraction for e-environment. In: *Proc. European conf. of the Czech Presidency of the Council of the EU towards eEnvironment*, 2009, s. 138–143.

Dill S., Eiron N., Gibson D., Gruhl D., Guha R., Jhingran A., Kanungo T., Rajagopalan S., Tomkins A., Tomlin J. A., Zien J. Y.: Semtag and Seeker: Bootstrapping the semantic web via automated semantic annotation. In: *Proc. The 12th International World Wide Web Conference (WWW)*. Budapest, 2003.

Embley D. W.: Toward Semantic Understanding – An Approach Based on Information Extraction Ontologies. In: *Proc. Conferences in Research and Practice in Information Technology*, 2004.

Embley D. W., Tao C., Liddle S. W.: Automatically extracting ontologically specified data from HTML tables with unknown structure. In: *Proc. ER*, 2002.

Etzioni O., Banko M., Soderland S. Weld D. S.: *Open Information Extraction from the Web. Communications of the ACM*, 2008.

Etzioni O., Cafarella M., Downey D., Kok S., Popescu A. M., Shaked T., Soderland S., Weld D. S., Yates A.: Web-scale information extraction in Knowitall: (preliminary results). In: *Proc. The 13th international conference on World Wide Web*, 2004.

Ferrucci D., Brown E., Chu-Carroll J., Fan J., Gondek D., Kalyanpur A. A., Lally A., Murdock J. W., Nyberg E., Prager J., Schlaefer N., Welty C.: Building Watson: An Overview of the DeepQA Project. In: *AI Magazine, AAAI*, 2010.

Finn A.: *A Multi-Level Boundary Classification Approach to Information Extraction*. Phd thesis, University College Dublin, 2006.

Freitag D., McCallum A.: HMM Structures Learned by Stochastic Optimization. In: *Proc. The 17th National Conference on Artificial Intelligence (AAAI)*, 2000.

Getoor L., Taskar B.: *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

Hajič J.: *Complex Corpus Annotation: The Prague Dependency Treebank*. Jazykovědný ústav L. Štúra, SAV, Bratislava 2004.

Hajič J., Holub M., Hučínová M., Pavlík M., Pecina P., Straňák P., Šidák P. M.: Validating and Improving the Czech WordNet via Lexico-Semantic Annotation of the Prague Dependency Treebank. In: *Proc. 4th Intl. Conf. on Language Resources and Evaluation, Workshop on Building Lexical Resources from Semantically Annotated Corpora*, 2004.

Handschuh S., Staab S.: Authoring and Annotation of Web Pages in CREAM. In: *Proc. The 11th International World Wide Web Conference (WWW)*, 2002.

Handschuh S., Staab S., Ciravegna F.: S-CREAM – Semi-automatic CREAtion of Metadata. In: *Proc. EKAW*, 2002.

Hastie T., Tibshirani R., Friedman J. H.: *The Elements of Statistical Learning*. 2. vydání, Springer, 2009.

Hepp M.: GoodRelations: An Ontology for Describing Products and Services Offers on the Web. In: *Proc. EKAW 2008*, Springer LNCS, vol. 5268, s. 332–347.

Hepp M., Siorpaes K., Bachlechner D.: Towards the Semantic Web in ETourism: Can Annotation Do the Trick. In: *Proc. The 14th European Conference on Information Systems (ECIS)*, 2006.

Hsu C.: Initial Results on Wrapping Semistructured Web Pages with Finite-State Transducers and Contextual Rules. In: *Proc. AAAI-98 Workshop on AI and Information Integration*, 1998.

Chakravarthy A., Ciravegna F., Lanfranchi V.: Cross-media document annotation and enrichment. In: *Proc. The 1st Semantic Authoring and Annotation Workshop (SAAW)*, 2006.

Chang C. H., Kayed M., Girgiz M. R., Shaalan K. F.: A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering*, 2006.

Jelinek F.: *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, Massachusetts, 1997.

Jurčíček F., Gašić M., Keizer S., Mairesse F., Thomson B., Yu K., Young S.: Transformation-based Learning for Semantic parsing. In: *Proc. Interspeech*, 2009.

Karkaletsis V., Karampiperis P., Stamatakis K., Labský M., Růžička M., Svátek V., Mayer M. A., Leis A., Villarroel D.: Automating Accreditation of Medical Web Content. In: *Proc. 18th European Conference on Artificial Intelligence (ECAI)*, 2008.

Kiryakov A., Popov B., Terziev I., Manov D., Ognyanoff D.: Semantic annotation, indexing, and retrieval, *Journal of Web Semantics*, vol. 2, 2004, s. 49–79.

Koller D., Friedman N.: *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009.

Kravalová J., Žabokrtský Z.: Czech Named Entity Corpus and SVM-based Recognizer. In: *Proc. Named Entities Workshop: Shared Task on Transliteration*, 2009.

Kunc M., Burget R.: Klasifikace prvků dokumentu na základě vizuálních rysů. In: *Sborník konference Znalosti*, 2008.

Kushmerick N., Weld D. S., Doorenbos R.: Wrapper Induction for Information Extraction. In: *Proc. International Joint Conference on Artificial Intelligence*, 1997.

Labský M., Svátek V.: Combining Multiple Sources of Evidence in Web Information Extraction. In: *Proc. ISMIS 17th Intl. Conf. on Foundations of intelligent systems*. Springer-Verlag, 2008, s. 471–476.

Labský M., Svátek V., Šváb O., Praks P., Krátký M., Snášel V.: Information Extraction from HTML Product Catalogues: From Source Code and Images to RDF. In: *IEEE/WIC/ACM Intl. Conference on Web Intelligence*, 2005, s. 401–404.

Lafferty J., McCallum A., Pereira F.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In: *Proc. The 18th International Conf. on Machine Learning*, 2001, s. 282–289.

Lerman K., Minton S. N., Knoblock C. A.: Wrapper Maintenance: A Machine Learning Approach. In: *Journal of Artificial Intelligence Research*, vol. 18, 2003.

Manning C. D., Schuetze H.: *Foundations of Statistical Natural Language Processing*, 2. vydání. The MIT Press, 2000.

McCallum A., Freitag D., Pereira F.: *Maximum Entropy Markov Models for Information Extraction and Segmentation*. In: Proc. ICML, 2000.

Moens M. F.: *Information Extraction: Algorithms and Prospects in a Retrieval Context*. Springer, 2006.

Muslea I., Minton S., Knoblock C.: Hierarchical Wrapper Induction for Semistructured Sources. In: *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 4, 2001, s. 93–114.

Peshkin L., Pfeffer A.: Bayesian Information Extraction Network. In: *Proc. Intl. Joint Conference on Artificial Intelligence*, 2003.

Petasis G., Karkaletsis V., Paliouras G., Androutsopoulos I., Spyropoulos C. D.: Ellogon: A New Text Engineering Platform. In: *Proc. 3rd International Conference on Language Resources and Evaluation (LREC)*, 2002.

Rabiner L. R.: A tutorial on hidden Markov models and selected applications in speech recognition. In: *Proc. IEEE*, 1989.

Richardson M., Domingos P.: Markov Logic Networks. In: *Machine Learning Journal*, 2006.

Roth D., Yih W.: Probabilistic Reasoning for Entity and Relation Recognition. In: *Proc. Intl. Conf. on Computational Linguistics (COLING)*, 2002.

Smrž P.: Quality Control for Wordnet Development. In: *Proc. 2nd Intl. WordNet Conference*, 2004.

Soderland S.: Learning information extraction rules for semi-structured and free text. In: *Machine learning*, vol. 34 (1/3), 1999, s. 233–272.

Ševčíková M., Žabokrtský Z., Krůza O.: *Zpracování pojmenovaných entit v českých textech*. Technická zpráva ÚFAL/CKL TR-2007-36, 2007.

Takeuchi K., Collier N.: Use of Support Vector Machines in Extended Named Entity Recognition. In: *Proc. CoNLL*, 2002.

Uren V., Cimiano P., Iria J., Handschuh S., Vargas-Vera M., Motta E., Ciravegna F.: *Semantic annotation for knowledge management: Requirements and a survey of the state of the art*. Web Semantics, 2006.

Vilain M., Burger J., Aberdeen J., Connolly D., Hirschman L.: A model-theoretic coreference scoring scheme. In: *Proc. 6th conf. on Message understanding*, 1995.

Weizenbaum J.: *ELIZA:* A computer program for the study of natural language communication between man and machine, *Comms. of the ACM*, vol. 9, Issue 1, 1966.

Witten I. H., Frank E.: *Data Mining: Practical machine learning tools and techniques.* 2. vyd., Morgan Kaufmann, San Francisco, 2005.

Žabokrtský Z., Ptáček J., Pajas P.: TectoMT: Highly Modular MT System with Tectogrammatics Used as Transfer Layer. In: *Proc. 3rd Workshop on Statistical Machine Translation*, 2008.

7 Automatické uvažování

V roce 1933 našel matematik Edward Huntington elegantní axiomatizaci jedné ze základních struktur používaných v matematické logice: Booleovy algebry. Krátce nato jiný matematik, Herbert Robbins, navrhl další zjednodušení této axiomatizace. Bylo zřejmé, že každá Booleova algebra tyto zjednodušené Robbinsovy axiomy splňuje, avšak dokázat že Robbinsovy axiomy jsou pro charakterizaci Booleových algeber dostačující, se ukázalo být těžké. Huntington, Robbins a další přední logici, jako Alfred Tarski, tento problém nedokázali vyřešit, takže zůstal otevřený po několik dalších desetiletí. Až v roce 1996 byl tento problém kladně vyřešen programem EQP (automatickým dokazovačem vět vyvinutým Williamem McCunem), který tehdy po osmi dnech výpočtů nalezl důkaz, že Robbinsovy axiomy jsou postačující. Správnost tohoto důkazu byla krok po kroku ověřena dalšími nezávislými systémy a důkaz byl zjednodušen do lidsky přístupné formy a publikován v *Journal of Algebra* (Dahn, 1998).

7.1 Co je automatické dokazování vět

Automatické dokazování vět (Automated Theorem Proving, ATP) je obor na pomezí umělé inteligence a formální matematiky zabývající se vývojem a implementací algoritmů ukazujících, že zadané matematické tvrzení (*domněnka*, conjecture) je logickým důsledkem zadané množiny *axiomů*. Automatické dokazovače se dnes používají v řadě oborů. Matematici mohou dokazovače používat jak k rychlému formálnímu ověřování jednoduchých domněnek (například že operace inverze v grupě je involucí), tak k těžkým novým důkazům, jako je uvedená Robbinsova domněnka. Vývojáři hardwaru a softwaru mohou automatické dokazování použít pro částečnou nebo úplnou formální kontrolu správnosti kritických částí svých návrhů, pokud tyto návrhy formálně popíší matematickými axiomy a požadovanou funkčnost vyjádří jako matematickou domněnku. Automatické dokazování lze použít k řešení logických problémů a hlavolamů, od sudoku přes rozestavování dam na šachovnici tak, aby se neohrožovaly, až po problém složení Rubikovy kostky. Tyto a podobné úlohy se dají řešit automatickým dokazovačem, pokud se problémy správně popíší pomocí axiomů a domněnek.

Jazykem (a *formalismem*) používaným pro formulaci problémů ATP je nejčastěji klasická predikátová logika (logika prvního řádu). Vyvíjejí se však také dokazovače jak pro výrazově silnější logiky (logiky vyšších řádů, neklasické logiky, modální logiky), tak pro výrazově slabší formalismy, jako je rovnostní logika, nebo dokonce čistá výroková logika a její různá zesílení. Použití přesného matematického formalismu v zápisu problémů je nutné pro práci dnešních dokazovačů. Na rozdíl od přirozeného jazyka (jako je čeština nebo angličtina) je matematická formulace jednoznačná a má přesně definovaný význam. Matematický formalismus také přesně definuje, jaká pravidla je možné použít pro odvozování nových tvrzení z axiomů, a tím i vymezuje pojem *korekt-*

ního důkazu. Převod problému z přirozeného jazyka do formálního matematického jazyka (*formalizace* nebo *matematizace* problému) je často netriviálním procesem, který ale na druhou stranu také často přispívá k lepšímu porozumění problému.

Výsledkem práce automatických dokazovačů je obvykle detailní *formální důkaz* domněnky z axiomů, tedy posloupnost odvozovacích kroků povolených daným matematickým formalismem. Správnost takového formálního důkazu lze tedy obvykle snadno nezávisle ověřit, ať už lidmi, nebo dalšími počítačovými programy (tzv. *ověřovači formálních důkazů*, formal proof checkers). To je také obecně jedna z velkých výhod formálních a počítači srozumitelných důkazů oproti důkazům nebo vysvětlením zapsaným v přirozeném jazyce. Není totiž neobvyklé, že lidé navrhnou a „pochopí" i důkazy, které jsou nesprávné nebo nedovysvětlené. Detailnost formálních důkazů ale může být i jejich nevýhodou, někdy je totiž těžké z dlouhé posloupnosti formálně korektních kroků vyextrahovat to, co lidé nazývají „podstatnými myšlenkami" daného důkazu, tedy důkaz lidsky pochopit. V některých případech lze z formálních důkazů také získat proceduru vedoucí k řešení nějakého problému. Například důkaz řešitelnosti uvedeného hlavolamu rozestavení dam na šachovnici by obvykle v sobě obsahoval popis takového rozestavení, kde se dámy neohrožují, a důkaz řešitelnosti problému složení Rubikovy kostky by v sobě obsahoval posloupnost kroků vedoucí z výchozí konfigurace do cílové.

Automatické dokazovače vět jsou tedy počítačové programy snažící se o řešení libovolného problému zapsaného v daném matematickém formalismu, obvykle logice prvního řádu. Logika prvního řádu je ale velmi silný výrazový prostředek, který umožňuje popsat v podstatě veškerou matematiku. Od Gödelových, Churcheových a Turingových dob (30. léta 20. století) je známo, že tato výrazová síla není zadarmo. Zjednodušeně řečeno, cenou za velkou výrazovou sílu je neexistence obecné efektivní procedury pro řešení problémů zapsaných v logice prvního řádu. Řečeno přesněji, logika prvního řádu není *rozhodnutelná* (decidable), je pouze *semirozhodnutelná* (semidecidable). Tyto pojmy znamenají, že neexistuje algoritmus, který pro každý problém logiky prvního řádu zaručeně někdy skončí a oznámí výsledek, tedy zda příslušná domněnka je nebo není logickým důsledkem příslušných axiomů. Lze pouze najít algoritmus (a takové algoritmy často používají systémy ATP), který vždy skončí úspěchem, pokud je domněnka skutečně důsledkem axiomů, ale který nemusí nikdy skončit, pokud tomu tak není. V praxi to znamená, že pokud automatické dokazování trvá dlouho, je to buď proto, že důkaz existuje, ale je těžký, nebo proto, že důkaz vůbec neexistuje (i v tom případě může dokazovač často po nějaké době skončit neúspěchem, místo aby neskončil vůbec). Dokazovač můžeme nechat pracovat hodinu, den nebo rok, ale dokud neskončí, tak nevíme, o který z těchto případů jde.

Tyto vlastnosti automatického dokazování vedou k tomu, že systémy ATP k řešení netriviálních problémů v rozumném čase často vyžadují ovládání specialistou, který do nějaké míry rozumí jak řešené matematické problematice, tak způsobu práce daného dokazovače. Slovo automatické v názvu systémů ATP je tedy třeba brát s rezervou, protože tito specialisté obvykle používají systémy ATP do jisté míry *interaktivně*. Tato interakce může být na různých úrovních. Uživatel může dokazovači například detailně říkat, která odvozovací pravidla se mají použít. Nebo se naopak uživatel vůbec nemusí o způsob práce dokazovače zajímat, pouze může daný problém rozdělit na menší podproblémy (lemmata), o kterých ví, že budou pro celkový důkaz potřebné. Systémy ATP a jejich uživatelé se tedy během procesu dokazování často navzájem ovlivňují a doplňují:

- systémy potřebují přesný popis problému zapsaný v logickém formalismu,
- to nutí uživatele k dodatečnému myšlení o tom, jak problém formálně vystihnout, což obvykle vede k hlubšímu porozumění problému,
- systém se snaží formalizovaný problém řešit,
- pokud se to podaří, dá systém uživateli k dispozici důkaz, který se uživatel může snažit pochopit, což může vést k tomu, že si uživatel uvědomí chybu ve formalizaci problému,
- pokud se problém nepodaří vyřešit, uživatel se může snažit dokazovači nějak pomoci, např. volbou různých nastavení, nebo se může snažit získat důkaz nějakého snadnějšího lemmatu a popřípadě i znovu zkontrolovat, zda je problém správně formálně popsán,
- a takto se celý proces interakce opakuje.

ATP je tedy jako technologie velmi vhodná tehdy, když specialista v nějakém oboru dokáže jasně a logicky formulovat problémy a interagovat se silnými automatickými systémy, které tyto problémy řeší. Ačkoliv tato kapitola vysvětluje základní principy práce systémů ATP, není nutné, aby se potenciální uživatelé ihned začali zabývat otázkou, jak takový systém ATP implementovat. Volně k dispozici je totiž řada dobrých automatických dokazovačů, se kterými lze dnes dokázat opravdu mnoho. Pro automatické dokazování v logice prvního řádu to jsou například systémy SPASS (Weidenbach, 2009), E (Schulz, 2002), Vampire (Riazanov, Voronkov, 2001), Prover9, Mace4, Paradox (Claessen, Sörensson, 2003), Z3 (De Moura, Bjørner, 2008), iProver (Korovin, 2008), Otter (McCune, 2003) a Waldmeister pro čistě rovnostní logiku (Hillenbrand, 1997). Pro (často do nějaké míry interaktivní) dokazování v logikách vyššího řádu jsou jimi například systémy HOL Light (Harrison, 2009), Isabelle (Nipkow, 2002), PVS (Owre, Shankar, 2008), ACL2 (Kaufmann, Moore, 2011), Coq (Bertot, Castéran, 2004), LEO (Benzmüller, 2008), Satallax (Brown, 2012) a TPS (Andrews, Brown, 2006). Pro automatické dokazování v čistě výrokové logice jmenujme alespoň MiniSAT (Eén, Sörensson, 2003), Barcelogic (Nieuwenhuis, Oliveras, 2005) a zChaff (Mahajan, 2004). K dispozici je také velké množství problémů z různých oborů zapsaných ve formalismu vhodném pro automatické dokazovače. Pro dokazování v logice prvního řádu je významným zdrojem problémů *knihovna TPTP*[1] (Thousands of Problems for Theorem Provers). Tato knihovna má také užitečné webové rozhraní[2] (*SystemOnTPTP*), v němž lze snadno mnohé automatické dokazovače prvního řádu použít, a tím obejít potřebu jejich instalace na počítači uživatele.

Automatické dokazování je dnes používané v řadě oborů, jmenujme alespoň matematiku (logiku, algebru a řadu dalších matematických teorií) a informatiku (verifikaci hardwaru a softwaru, zpracování přirozeného jazyka a další). Nejvýznamnější aplikace jsou popsány v následujícím výkladu. Potenciální použití ATP je však mnohem širší, téměř pro jakýkoliv obor, který dosáhl alespoň částečné logické (sémantické) formulace svých znalostí. Díky rychle postupujícímu pronikání počítačové techniky do celé vědy a rozvoji velkých znalostních bází veřejně přístupných na internetu tak ATP začíná pronikat i do oborů, jako je biologie, chemie, medicína, obchod atd. Dá se říci, že technologie ATP v dnešní době čeká na své nové uživatele z těchto oborů.

[1] http://www.tptp.org
[2] http://www.tptp.org/cgi-bin/SystemOnTPTP

7.2 K čemu je automatické dokazování vět

Patrně nejznámějším automaticky dokázaným matematickým výsledkem (oznámeným například i v New York Times) je již uvedený důkaz Robbinsovy domněnky. Automatické dokazovače jsou často používány skupinou matematiků pracujících v teorii kvazigrup, která je soustředěna kolem dokazovače Prover9 (následník systémů Otter a také EQP, vyvíjený Williamem McCunem). Této skupině (William McCune, Ken Kunen, Michael Kinyon, Bob Veroff, J. D. Phillips a také David Stanovský z University Karlovy) se již podařilo automaticky dokázat řadu nových významných tvrzení v teorii kvazigrup (Phillips, Stanovský, 2010). William McCune spolu s kolegy použil svůj systém Otter k hledání minimálních axiomatik, nových jednoprvkových axiomatizací a řešení dalších zajímavých problémů v řadě algebraických struktur[3]. Výkonný rovnostní dokazovač Waldmeister byl nedávno (2007) integrován do komerčního programu Mathematica používaného patrně na většině univerzit na světě.

7.2.1 Použití v rámci interaktivních dokazovačů

V nedávné době se také začalo rozvíjet použití ATP v rámci *větších interaktivních dokazovačů*. První formálně ověřený důkaz Jordanovy věty („Rovinná uzavřená křivka, která sama sebe nikde neprotíná, rozděluje rovinu na dvě části – vnitřek a vnějšek křivky.") byl sestrojen teprve v roce 2005 matematikem Thomasem Halesem v interaktivním systému HOL Light vyvíjeným Johnem Harrisonem z Intelu. Tento velký interaktivně vytvořený formální důkaz z rovinné topologie má zhruba 60 000 řádek a obsahuje zhruba 3000 volání automatického dokazovače zabudovaného do systému HOL Light. Bez možnosti použití ATP by dokazování takového počtu lemmat zabralo mnohem více úsilí a podstatně prodloužilo tento významný projekt, který už i tak trval špičkovému matematikovi (Hales v roce 1998 dokázal Keplerovu domněnku) deset měsíců. Automatické dokazovače SPASS, E a Vampire byly v roce 2006 Larry Paulsonem integrovány také s dalším významným interaktivním dokazovačem Isabelle a toto propojení se stalo od té doby jednou z nejpoužívanějších dokazovacích metod v Isabelle. Patrně největší současná formální matematická knihovna vyvíjená systémem Mizar byla Josefem Urbanem převedena do logiky prvního řádu a systémy ATP byly použity k nezávislému ověření formálních Mizarových důkazů (Urban, Sutcliffe, 2007) a asistenci při jejich tvorbě a zjednodušování (Urban, 2006a), (Urban, 2006b).

7.2.2 Tvorba a verifikace softwaru

Tvorba a verifikace softwaru jsou ekonomicky významné aplikace automatického dokazování, které se pomalu stávají více a více používanými. V roce 2005 zveřejnil Tony Hoare (jeden ze zakladatelů formálního uvažování o programech a tvůrce tzv. Hoarovy logiky) „velkou výzvu" (Grand Challenge, obdoba například úspěšné dřívější výzvy k prozkoumání lidského genomu) softwarové verifikace. V ní vyzývá k nahrazení současných počítačových programů programy, jejichž správnost je formálně ověřena a kterou zdůvodňuje mimo jiné i ekonomickými argumenty. V roce 1999 mezinárodní

[3] http://www-unix.mcs.anl.gov/AR/new_results/

standardizační organizace (ISO a IEC) vytvořily *Obecná kritéria pro bezpečnost informačních technologií* (Common Criteria for Information Technology Security Evaluation - standard ISO/IEC 15408), která v sobě obsahují popis sedmi úrovní hodnocení bezpečnosti softwaru podle těchto kritérií, tzv. *Evaluation Assurance Levels*. Tři nejvyšší úrovně (EAL5, EAL6 a EAL7) vyžadují částečnou nebo úplnou formální verifikaci. V říjnu 2008 byl americkou firmou Praxis dokončen a volně zveřejněn jeden z prvních projektů (Tokeneer[4]) plně splňujících podmínky EAL5, který si u dotyčné firmy (pro demonstraci těchto postupů) objednala americká Národní bezpečnostní agentura (NSA). Tento projekt využívá jazyk SPARK, což je verze jazyka ADA zjednodušená a upravená pro účely formální verifikace. Takové formální ověření správnosti programu je velmi podobné formálnímu ověření matematických vět (jako je například zmíněná Jordanova věta), a tedy i v tomto případě mohou automatické dokazovače podstatně usnadnit práci. Vedle firmy Praxis a jejích projektů existuje řada dalších firem a projektů zabývajících se dnes formální verifikací softwaru. Například v Microsoftu je vyvíjený programovací systém Spec#[5], který rozšiřuje jazyk C# o konstrukce potřebné k formální specifikaci a verifikaci. Tento systém je propojen s automatickým dokazovačem, který verifikační podmínky analyzuje a snaží se dokázat korektnost popsaného programu, nebo v něm najít chyby.

V americkém Národním úřadu pro letectví a kosmonautiku (NASA) je kromě verifikace softwaru také několik projektů zabývajících se přímou automatickou konstrukcí softwaru. Projekt Amphion[6] používá automatické dokazovače k deduktivní konstrukci programů ze softwarových komponent. Tento projekt (a vývojové paradigma) lze použít pro řadu aplikací, jednou z nich byla například konstrukce softwaru potřebného pro planetární mise prováděné v NASA. Projekt „Certifikovatelná syntéza" (Certifiable Synthesis) jde v tomto směru v NASA ještě dál a generuje kód navíc s anotacemi, které lze použít pro automatické dokazování různých (obvykle bezpečnostních) vlastností generovaných programů.

7.2.3 Verifikace hardwaru

Verifikace hardwaru je dnes patrně největší průmyslovou aplikací ATP. Firmy jako Intel, AMD, IBM a Motorola používají technologii ATP pro formální verifikaci. Například zmíněný systém HOL Light je v Intelu používán Johnem Harrisonem k verifikaci různých hardwarových řešení, třeba floating-point aritmetiky procesorů (Harrison, 2007). Nemalým impulsem k rozvoji použití formálních metod ve velkých hardwarových firmách byla právě chyba v aritmetice floating-point procesoru Intel Pentium nalezená v roce 1994, která nakonec donutila Intel k (potenciálně velmi drahé) nabídce výměny všech postižených procesorů. Obdobně jako systém HOL Light byl systém ACL2 použit k verifikaci aritmetiky floating-point procesorů AMD a k verifikaci různých vlastností procesorů ARM. Systém PVS byl použit k verifikaci procesorů sloužících k řízení letu letadel a systém Nqthm ke kompletnímu verifikovanému návrhu procesoru FM9001.

[4] http://www.adacore.com/home/gnatpro/tokeneer/
[5] http://research.microsoft.com/en-us/projects/specsharp/
[6] http://ti.arc.nasa.gov/project/pro-amphion/

7.3 Potřebné pojmy z matematické logiky

Budeme předpokládat znalost běžných pojmů z predikátové logiky, jakými jsou: *proměnná*, *konstanta*, *funkční symbol*, *term*, *predikátový symbol*, *logická spojka*, *kvantifikátor*, *formule*, *atomická formule*, *otevřená a uzavřená formule*, *sentence*, *podterm*, *podformule*, *substituce* a *unifikace*. Blíže jsou tyto pojmy vysvětleny například v kapitole 3 prvního dílu Umělé inteligence. *Literálem* nazveme atomickou formuli nebo její negaci. Dva literály nazveme *komplementární*, pokud jeden je negací druhého. *Klauzulí* nazveme konečnou množinu literálů. Obvyklé chápání klauzulí je disjunktivní, tedy jako disjunkce literálů, a tak tomu bude i v této kapitole. Tedy například množina literálů $\{\neg C, A, \neg B\}$ je klauzule reprezentující disjunkci literálů $\neg C, A, \neg B$. *Prázdná klauzule*, tedy prázdná množina literálů, je při tomto disjunktivním chápání vždy nepravdivá. Všechny proměnné v literálech klauzule jsou implicitně univerzálně kvantifikované. *Jazykem* rozumíme množinu predikátových a funkčních symbolů, spolu s jejich aritami, a *teorií* v daném jazyce rozumíme množinu sentencí v tomto jazyce. Předpokládáme znalost pojmů, jako je *interpretace jazyka*, *model teorie* a *splnitelnost a pravdivost formule v modelu*. Pravdivost formule A v modelu M značíme $M \models A$, a také říkáme že M je modelem formule A. Množina formulí T je pravdivá v modelu M (nebo také M je modelem T), značíme $M \models T$, pokud je v M pravdivá každá formule z T. Formuli (klauzuli) nazýváme *tautologií*, pokud je pravdivá v každé interpretaci svého jazyka. Formuli A nazveme *logickým důsledkem* množiny formulí T, značíme $T \models A$, pokud každý model množiny formulí T je také modelem formule A. *Odvozovacím pravidlem* (inference rule) budeme nazývat jakékoliv pravidlo pro odvození logických formulí (*důsledků*) z žádné nebo více formulí (*předpokladů*). Známým příkladem odvozovacího pravidla je *modus ponens*, které ze dvou předpokladů tvaru $A \rightarrow B$ a A odvodí důsledek B:

$$\frac{A \rightarrow B, \quad A}{B}.$$

Řekneme, že odvozovací pravidlo I je *korektní* (correct nebo sound), pokud formule odvozená pomocí I je vždy logickým důsledkem svých předpokladů, neboli pokud $T \vdash_I A$, pak také $T \models A$ pro všechny množiny předpokladů T a jejich důsledky A. Řečeno ve zkratce, korektnost znamená, že co lze odvodit, to je pravda. Pokud jsou již zadána nějaká odvozovací pravidla, řekneme, že *formule A je dokazatelná z množiny formulí T* (pomocí povolených odvozovacích pravidel), pokud existuje konečná posloupnost formulí $A_1, \ldots, A_n$ taková, že $A_n = A$ a každá formule A_i je buď prvkem T, nebo je odvozena nějakým povoleným odvozovacím pravidlem z předpokladů, které jsou v $T \cup \{A_j : j < i\}$. To značíme $T \vdash A$ a posloupnosti formulí A_i říkáme *důkaz formule A* z množiny formulí T. Řekneme, že *množina odvozovacích pravidel je úplná* (complete), pokud pro každou množinu sentencí T a sentenci A takové, že $T \models A$, platí také, že $T \vdash A$. Řečeno ve zkratce, úplnost znamená, že každý logický důsledek je také možné dokázat.

Dříve než můžeme provádět automatické dokazování, je potřeba vyjádřit daný problém v jazyce srozumitelném počítači. To se nazývá *formalizací problému*. V následujícím příkladu ukážeme formalizaci jednoduchého logického problému v predikátové logice.

Zadejme problém v přirozeném jazyce[7]: Máme zjistit, zda Johnovi chutnají buráky (C). Víme, že mu chutná každé jídlo (A7) a že jablka a kuře jsou jídlo (A1). Dále víme, že Bill je naživu (A3), a kdo je naživu, toho nic nezabilo (A6). Bill jí buráky (A4) a co jí Bill, to jí také Sue (A5). Pokud někdo něco jí a nezabije ho to, pak je to jídlo (A2).

Nyní ukažme formalizaci problému:

% conjecture C:
likes(john,peanuts)

% axiom A1:
food(apples) & food(chicken)

% axiom A2:
$(\forall X : (\exists Y : \text{eats}(Y,X) \ \&\ \text{not_killed_by}(Y,X)) \rightarrow \text{food}(X))$

% axiom A3:
alive(bill)

% axiom A4:
eats(bill,peanuts)

% axiom A5:
$(\forall X : \text{eats}(\text{bill},X) \rightarrow \text{eats}(\text{sue},X))$

% axiom A6:
$(\forall Y : \text{alive}(Y) \rightarrow (\forall X : \text{not_killed_by}(Y,X)))$

% axiom A7:
$(\forall X : \text{food}(X) \rightarrow \text{likes}(\text{john},X))$

Přirozeně je možné problémy formalizovat různými způsoby, ale musíme počítat s tím, že ne každá formalizace je pro počítačové odvozování stejně vhodná. Často se například stává, že problém obsahuje nadbytečné axiomy, tj. formule, které ve skutečnosti nejsou k řešení problému potřeba (zkuste se zamyslet, které to jsou v našem logickém problému). Jak uvidíme později, takové nadbytečné axiomy obvykle způsobují zvětšení prohledávacího prostoru, a tedy obvykle zpomalí počítačové odvozování.

[7] V závorkách za každou větou jsou uvedeny názvy formulí predikátové logiky, které jim odpovídají.

7.4 Rezoluční automatické dokazování

V tomto oddílu a dále pak v oddílu 7.5 rozebereme dva nejběžnější způsoby práce automatických dokazovačů: rezoluční metodu a metodu tableau. Nadále se (pokud nenapíšeme něco jiného) budeme zabývat dokazováním v klasické predikátové logice. Budeme předpokládat, že problém, který má dokazovač vyřešit, je zadán jako množina axiomů (označme ji Ax) a jako domněnka (označme ji C, jako conjecture), která má být z těchto axiomů dokázána.

7.4.1 Rezoluční metoda

Dnes nejobvyklejší metodou automatického dokazování je *rezoluční metoda – klauzální rezoluční zamítací dokazování* s tzv. *smyčkou ANL*. Vysvětleme si nyní tyto pojmy podrobněji.

Klauzálním dokazováním (clausal theorem proving) se rozumí to, že dokazovač pracuje (tedy například provádí logické inference) pouze se speciálním tvarem logických formulí – *klauzulemi*. Axiomy a domněnka jsou na začátku převedeny do klauzulí programem (ten může být součástí dokazovače), který se nazývá *klauzifikátor* (clausifier). Klauzifikace je stručně popsána v kapitole 3 prvního dílu Umělé inteligence. Tento proces může obsahovat řadu optimalizací, které mají vliv na tvar výsledných klauzulí, řada z nich je popsána v knize (Nonnengart, Weidenbach, 2001). V obecném případě se nelze vyhnout rozšiřování původního jazyka zaváděním tzv. skolemovských konstant a funkcí, které umožňují odstraňovat existenční kvantifikátory z formulí. Všechna odvozovací pravidla dokazovače jsou pak formulována pouze pro klauzule.

Zamítacím dokazováním (refutational theorem proving) rozumíme způsob dokazování, kdy nejprve znegujeme dokazovanou domněnku C a spolu s axiomy Ax se pak z množiny $Ax \cup \{\neg C\}$ snažíme dokázat spor (obvykle tak, že odvodíme prázdnou klauzuli). Je poměrně častým jevem, že odvozovací pravidla používaná dokazovačem nejsou úplná, avšak jsou *refutačně úplná* (refutationally complete). Tedy odvozovací pravidla nejsou obvykle dostatečně silná na to, aby, pokud platí $Ax \models C$, dokazovač přímo odvodil C z Ax, ale stačí na to, aby v tomto případě odvodil z množiny $Ax \cup \{\neg C\}$ spor. Klauzální dokazovače mají obvykle jako vstup své hlavní procedury množinu klauzulí, jejíž spornost se snaží dokázat, a rozdělení tohoto vstupu na klauzule odpovídající axiomům a klauzule odpovídající negované domněnce je pro dokazovač pouze heuristickou informací, kterou může, ale nemusí využít.

Rezolučním dokazováním (resolution theorem proving) je dokazování, jehož základním odvozovacím pravidlem je *rezoluce*. Rezoluční kalkulus představil v roce 1965 matematik J. A. Robinson. Tento kalkulus v nejjednodušší podobě má pouze jedno odvozovací pravidlo (rezoluci), nemá žádné axiomy a pracuje pouze s klauzulemi. Pro snadné pochopení je dobré si zapamatovat speciální binární výrokovou verzi (zvanou také *pravidlo řezu* (cut rule)): Jsou-li dány dvě klauzule $\{A, C\}$ a $\{\neg C, B\}$, pak tato verze umožňuje odvodit klauzuli $\{A, B\}$. Uvědomme si, že $\{A, C\}$ je totéž co $A \vee C$, což je totéž jako $\neg A \rightarrow C$. Obdobně $\{\neg C, B\}$ je to totéž co $C \rightarrow B$ a $\{A, B\}$ je totéž co $\neg A \rightarrow B$. V této formě jde tedy o poměrně intuitivní pravidlo zřetězení implikací (z $\neg A \rightarrow C$ a $C \rightarrow B$ odvozujeme $\neg A \rightarrow B$). Obecná predikátová verze pravidla *binární rezoluce* je

$$\frac{\{A_1,\ldots,A_m,C\},\quad \{B_1,\ldots,B_n,\neg D\}}{\{A_1,\ldots,A_m,B_1,\ldots,B_n\}\theta},$$

kde $A_1,\ldots,A_m$ a $B_1,\ldots,B_n$ jsou literály a θ je *unifikační substituce* θ literálů C a D (tedy taková nejobecnější substituce, že $C\theta \equiv D\theta$). Tak, jak je obvyklé při unifikaci, předpokládáme, že vstupní klauzule spolu nemají žádné společné proměnné.

Výsledná klauzule se nazývá *rezolventa* (resolvent) svých předpokladů, také někdy nazývaných *rodiče – rodičovské klauzule* (parent clauses). Literály C a $\neg D$ se nazývají *rezolvované literály* (literals resolved upon). Snadno se ověří, že rezoluce je korektním odvozovacím pravidlem, tedy že rezolventa je vždy logickým důsledkem svých rodičů.

Příklad. Zde vidíme všechny možné aplikace pravidla rezoluce na dvě klauzule: {silny(petr), silny(otec(petr))}, {¬silny(X), ¬silny(Y), vetsi(X,Y)}, kde X a Y jsou proměnné, petr/0 je konstanta, silny/1 je unární predikát, vetsi/2 je binární predikát a otec/1 je unární funktor.

1. možnost:
C = silny(petr) $\quad\quad \neg D = \neg$silny(X)
$\theta = \{X/\text{petr}\}$
rezolventa: {silny(otec(petr)), ¬silny(Y), vetsi(petr,Y)}

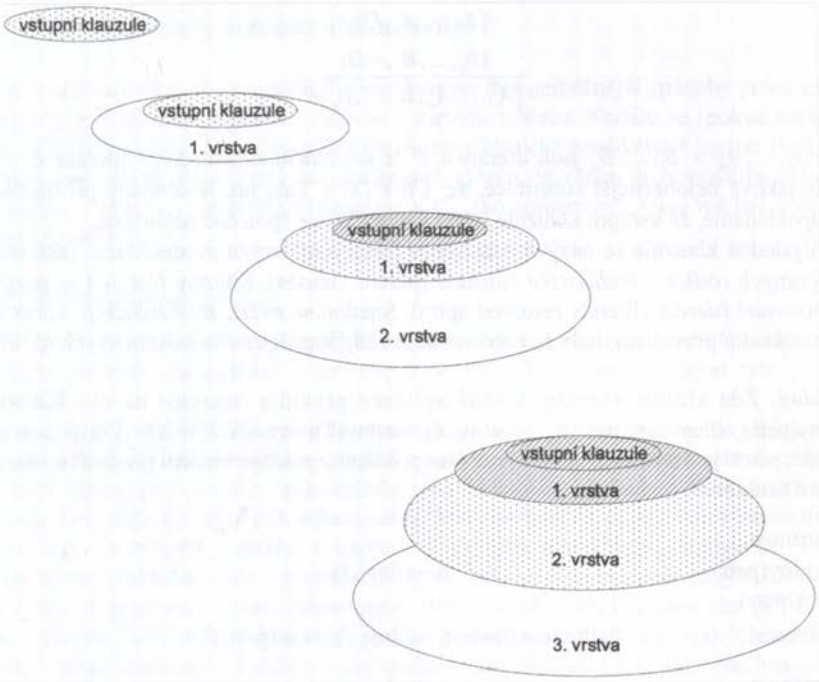
2. možnost:
C = silny(otec(petr)) $\quad\quad \neg D = \neg$silny(X)
$\theta = \{X/\text{otec(petr)}\}$
rezolventa: {silny(petr), ¬silny(Y), vetsi(otec(petr),Y)}

3. možnost:
C = silny(petr) $\quad\quad \neg D = \neg$silny(Y)
$\theta = \{Y/\text{petr}\}$
rezolventa: {silny(otec(petr)), ¬silny(X), vetsi(X,petr)}

4. možnost:
C = silny(otec(petr)) $\quad\quad \neg D = \neg$silny(Y)
$\theta = \{Y/\text{otec(petr)}\}$
rezolventa: {silny(petr), ¬silny(X), vetsi(X,otec(petr))}

7.4.2 Smyčka ANL

Jak je patrné na předchozím příkladu, hlavním problémem rezolučního dokazování je výběr klauzulí a jejich literálů, na které se bude aplikovat pravidlo rezoluce. Rezoluce je třeba provádět tak, abychom žádný krok potřebný k dosažení sporu nevynechali a abychom žádný krok zbytečně neopakovali. Obrázek 7.1 znázorňuje jednoduchý způsob odvozování klauzulí ve vrstvách, kde každá vrstva obsahuje všechny nové rezolventy vytvořené z předchozích vrstev.



Obr. 7.1

V každé iteraci je vybrána jedna klauzule z tečkované vrstvy a ostatní klauzule jsou vybírány z tečkované nebo čárkované vrstvy. Rezolventy jsou uloženy do bílé vrstvy a připraveny do další iterace. Tímto postupem (tzv. prohledáváním do šířky) sice zaručíme, že žádný potřebný krok nevynecháme, ale počet generovaných klauzulí roste velice rychle – a to dvojitě exponenciálně vůči číslu vrstvy. Obvykle už pro relativně malý počet vstupních klauzulí tento jednoduchý přístup vede k rychlé kombinatorické explozi problému. Našim cílem tedy je provádět jenom ty rezoluční kroky, které nám umožní dojít co nejdříve ke sporu. K dalšímu snížení takto rychlého růstu klauzulí je potřeba:

– Sledovat, které rezolventy a kdy jsou generovány.
– Používat restriktivní strategie k zamezení vzniku určitých rezolvent. Běžným efektem takových strategií je vynucení delšího důkazu, ale s omezenějším výběrem možností a tím zmenšením prohledávacího prostoru.
– Používat strategie mazání nepotřebných klauzulí (tedy klauzulí, o kterých víme, že nebudou využity pro nalezení sporu).

Flexibilním algoritmem vyvinutým pro tento účel v americké Národní laboratoři v Argonne (Argonne National Laboratory – ANL) je takzvaná *smyčka ANL* (*ANL Loop*). Množina vygenerovaných klauzulí je v ní rozdělena na dvě části:

– množinu zpracovaných (worked off) klauzulí, mezi kterými už všechna možná odvození byla provedena,
– zbývající použitelné (usable) klauzule čekající na zpracování.

Uvedený algoritmus popisuje základní fungování smyčky ANL:

```
% WorkedOff označuje množinu klauzulí, které už byly zpracovány
WorkedOff := { };

% Usable označuje množinu doposud nezpracovaných klauzulí
Usable := všechny vstupní klauzule;

while ( refutace nenalezena ) and ( Usable ≠ { } ) do
begin
  Vyber klauzuli ChosenClause z množiny Usable;
  Přesuň klauzuli ChosenClause do WorkedOff;
  Proveď všechna možná odvození mezi ChosenClause a ostatními klauzulemi
z množiny WorkedOff;
  Přidej odvozené klauzule do množiny Usable;
end.
```

Způsob volby *ChosenClause* z množiny *Usable* (z množiny doposud nezpracovaných klauzulí) je určen *strategií prohledávání*. Ta může být:

- *do hloubky* (depth first search), která
  - za *ChosenClause* volí vždy naposledy vygenerovanou rezolventu,
  - nezaručuje úplné prohledání prostoru (může se zacyklit v nekonečné smyčce),
  - nezaručuje nalezení refutace v nejmenším počtu odvození;
- *do šířky* (breadth first search), která
  - za *ChosenClause* volí „nejstarší" rezolventu (první nezpracovanou),
  - zaručuje úplné prohledání prostoru,
  - zaručuje nalezení refutace v nejmenším počtu odvození,
  - implementuje uvedené prohledávání po vrstvách (*ply-by-ply search*);
- *dle „nejlepšího výběru"* (best first search), která předpokládá, že nějakým způsobem umíme porovnávat vhodnost klauzulí. To nám umožňuje
  - vybrat pro rezoluci „nejlepší" klauzuli jako *ChosenClause*.

Příklad. Příklad ukazuje několik počátečních iterací smyčky ANL s prohledáváním do šířky:

Množina vstupních klauzulí = {{p(b), r(*Y*), p(*Y*)},
{¬p(*S*), p(b)},
{¬p(b)},
{¬r(a)},
{¬r(c)}
}

WorkedOff := {}
Usable := Množina vstupních klauzulí (počáteční pořadí klauzulí je dáno uživatelem)

1. iterace:
Nyní vybíráme *ChosenClause* jako první klauzuli z *Usable*.
ChosenClause := {p(b), r(*Y*), p(*Y*)}
WorkedOff nyní obsahuje jen *ChosenClause* , dále

Usable = {{¬p(*S*), p(b)},
{¬p(b)},
{¬r(a)},
{¬r(c)}
}

Žádné rezoluce v množině *ChosenClause* nejsou nyní možné, tedy do množiny *Usable* nic nepřidáváme.

2. iterace:

ChosenClause := {¬p(*S*), p(b)}
\+ {p(b), r(*Y*), p(*Y*)} ⇒ {r(*Y*), p(*Y*), p(b)}
{p(b), r(*Y*), p(b)}
{r(b), p(b)}

WorkedOff = {{p(b), r(*Y*), p(*Y*)},
{¬p(*S*), p(b)}
}

Usable = {{¬p(b)},
{¬r(a)},
{¬r(c)},
{r(*Y*), p(*Y*), p(b)},
{p(b), r(*Y*), p(b)},
{r(b), p(b)}
}

Provedli jsme tedy všechny (tři) možné rezoluce mezi *ChosenClause* a klauzulemi z *WorkedOff* a rezolventy přidali do *Usable*. V dalších iteracích již stavy množin klauzulí *WorkedOff* a *Usable* neuvádíme.

3. iterace:

ChosenClause := {¬p(b)}
\+ {p(b), r(*Y*), p(*Y*)} ⇒ {r(*Y*), p(*Y*)}
{p(b), r(*Y*)}
{r(b)}
\+ {¬p(*S*), p(b)} ⇒ {¬p(*S*)}

4. iterace:

ChosenClause := {¬r(a)}
\+ {p(b), r(*Y*), p(*Y*)} ⇒ {p(b), p(a)}

5. iterace:

ChosenClause := {¬r(c)}
+ {p(b), r(*Y*), p(*Y*)} ⇒ {p(b), p(c)}

6. iterace:
ChosenClause := {r(*Y*), p(*Y*), p(b)}
+ {¬p(*S*), p(b)} ⇒ {r(*Y*), p(b), p(b)}
{r(*Y*), p(*Y*), p(b)}
{r(*Y*), p(b)}
{p(b), p(c)}
+ {¬p(b)} ⇒ {r(*Y*), p(b)}
{r(*Y*), p(*Y*)}
{r(b)}
+ {¬r(a)} ⇒ {p(a), p(b)}
+ {¬r(c)} ⇒ {p(c), p(b)}

7.4.3 Faktorizace a další pravidla

Binární rezoluce sama o sobě není refutačně úplná. Jedním z příkladů je známý *holičův paradox* (barber's paradox) tvrdící, že neexistuje žádný holič *B*, který by holil právě ty, kdo neholí sami sebe

$$\neg(\exists B : (\forall X : \text{shaves}(B, X) \Leftrightarrow \neg\text{shaves}(X, X))).$$

Po znegování a převedení do klauzálního tvaru vzniknou z této formule následující dvě klauzule obsahující novou skolemovskou konstantu barber nahrazující existenční proměnnou. Každá klauzule vyjadřuje jednu z implikací uvedené ekvivalence:

$$\{\text{shaves(barber},X), \text{shaves}(X,X)\},$$
$$\{\neg\text{shaves(barber},Y), \neg\text{shaves}(Y,Y)\}.$$

Binární rezoluci je zde možné provádět řadou způsobů, avšak všechny vedou pouze na neužitečné tautologie. Proto je třeba binární rezoluci doplnit dalším odvozovacím pravidlem, *faktorizací*, která umožňuje sloučit dva unifikující se literály do jednoho

$$\frac{\{A_1,\ldots,A_m,C,D\}}{\{A_1,\ldots,A_m,C\}\theta},$$

kde $A_1,\ldots,A_m$ a C, D jsou literály a θ je unifikační substituce θ literálů C a D (tedy taková nejobecnější substituce, že $C\theta \equiv D\theta$).

Pomocí faktorizace už se nám snadno podaří odvodit spor z předchozích klauzulí pro holičův paradox:

1. iterace:
ChosenClause := {shaves(barber, *X*), shaves(*X*, *X*)}
⇨ faktorizace {shaves(barber, *X*)}
2. iterace:

ChosenClause := {shaves(barber, *X*)}
(žádnou rezoluci ani faktorizaci nelze provést)

3. iterace:
ChosenClause := {¬shaves(barber, *Y*), ¬shaves(*Y*, *Y*)}
$+_{\text{rezoluce}}$ {shaves(barber,*X*)} ⇨ {¬shaves(barber,*Y*)}
(další rezoluce a faktorizace)

4. iterace:
ChosenClause := {¬shaves(barber,*Y*)}
$+_{\text{rezoluce}}$ {shaves(barber,*X*)} ⇨ {} (spor)

Smyčka ANL je často doplňována i jinými odvozovacími pravidly, například faktorizací, a také pravidly umožňujícími odmazávat nepotřebné klauzule (například tautologiemi). Použitelnost odvozovacích pravidel lze také omezovat pomocí různých dodatečných podmínek kladených na tvar klauzulí. Binární rezoluce a faktorizace zaručují refutační úplnost, zatímco další pravidla a případná omezení použitelnosti přispívají k větší efektivnosti odvozování. Při kombinování různých odvozovacích a odmazávacích pravidel a různých omezujících podmínek je třeba dbát na zachování úplnosti vzniklé odvozovací procedury. Detailní popis řady těchto pravidel a jejich kombinaci lze najít například v systému SPASS (Weidenbach, 2009).

Pro čtenáře znalé programovacího jazyka Prolog poznamenejme, že jádrem Prologu je také speciální (omezená) verze rezoluce. Tento jazyk je historicky s automatickým dokazováním úzce spjat a řada automatických dokazovačů je v tomto jazyce implementována. Nejjednodušším příkladem, který maximálně využívá mechanismů jazyka Prolog pro automatické dokazování, je dokazovač tableau LeanTAP, který probíráme v dalším textu.

7.4.4 Jazyk TPTP

Dříve než se pokusíme automaticky vyřešit nějaký problém, je třeba se seznámit se zápisem problémů používaných počítači. Pro počítač srozumitelný (formální) zápis problémů a důkazů v predikátové logice bylo vyvinuto několik syntaxí. Dnes nejpoužívanější takovou syntaxí je jazyk TPTP[8], který se také používá v nejznámější knihovně problémů pro automatické dokazovače, zvané *Thousands of Problems for Theorem Provers* (Sutcliffe, Suttner, 1998). Tato syntax je používána také při každoročním mistrovství světa v automatickém dokazování vět pořádaném od roku 1996 na konferenci CADE[9] (Conference on Autometed Deduction). Tato soutěž se nazývá CASC[10] – *The CADE ATP System Competition* (Sutcliffe, Suttner, 2006).

Formální syntax, jíž je jazyk TPTP, by především měla být snadno srozumitelná pro počítače. Zároveň však také chceme, aby byla čitelná pro lidi tak, aby umožňovala snadno zapsat podstatné informace. Vzhledem k používanosti Prologu v automatickém doka-

[8] Aktuální formální syntax lze najít na http://tptp.org/TPTP/SyntaxBNF.html
[9] http://www.cadeconference.org/
[10] http://tptp.org/CASC/

zování je dobré, aby tato syntax byla prologovská, což jazyk TPTP respektuje. Jak zápis problémů, tak zápis důkazů v TPTP, tedy sestává z prologovských klauzulí tvaru:

```
fof(food_is,axiom,(food(apples) & food(chicken))).
cnf(food_is,axiom,(food(apples) | ~food(chicken))).
```

První prologovská klauzule kóduje obecnou formuli predikátové logiky (first order formula – fof). Jde o ternární prologovský predikát `fof/3`, kde prvním argumentem je název formule (`food_is`), druhým argumentem je role, ve které formule vystupuje (obvykle axiom nebo domněnka (conjecture)), a třetím argumentem je vlastní formule. Druhá prologovská klauzule je podobná, avšak místo obecné formule predikátové logiky kóduje klauzuli predikátové logiky, a tedy místo prologovského predikátu `fof/3` používá predikát `cnf/3` (clause normal form). V případě potřeby je možné do tohoto zápisu přidat i další informace, například o tom, odkud daná formule nebo klauzule pochází (tj. například jak byla odvozena). To uvidíme později. Nyní zapišme náš uvedený logický problém v syntaxi TPTP:

```
fof(eaten_is_food,axiom,
( ! [X] :
            ( ? [Y] :
                ( eats(Y,X)
                & not_killed_by(Y,X) )
            => food(X) ))).

fof(bill_alive,axiom,
  alive(bill)).

fof(bill_eats_peanuts,axiom,
  eats(bill,peanuts)).

fof(sue_eats_bills,axiom,
( ! [X] :
            ( eats(bill,X)
            => eats(sue,X) ))).

fof(alive_not_killed_by,axiom,
 (! [Y] :
            ( alive(Y)
            => ! [X] : not_killed_by(Y,X) ))).

fof(john_likes_food,axiom,
( ! [X] :
            ( food(X)
            => likes(john,X) ))).

fof(prove_this,conjecture,
  likes(john,peanuts)).
```

7.4.5 Automatické řešení problému dokazovačem

Pro demonstraci řešení našeho problému použijeme automatický dokazovač Eprover (Schulz, 2002), vyvíjený od roku 1999 Stephanem Schulzem z Technické univerzity v Mnichově. Jde o rezoluční klauzální dokazovač, který už má v sobě zabudovaný vlastní klauzifikátor. Tento dokazovač je buď možné stáhnout z webové stránky autora[11], nebo je možné použít zmíněné webové rozhraní SystemOnTPTP, kde si lze řadu dokazovačů vyzkoušet. Tento dokazovač se na náš problém (zapsaný v souboru `john.tptp`) spustí například příkazem:

```
eproof --print-statistics -xAuto -tAuto --cpu-limit=10 --memory-limit=Auto --tptp3-format john.tptp .
```

Povšimněmě si v tomto příkazu hlavně parametrů cpu-limit (= 10) a memory-limit (= Auto). Jak už jsme napsali v úvodní části, automatické dokazovaní v predikátové logice je obecně nerozhodnutelný problém a dokazovač tedy nemusí při hledání důkazu nikdy skončit. Proto je obvyklé práci dokazovače omezovat limity na zdroje, tedy na čas a na použitou počítačovou paměť (která se obvykle rychle zaplňuje generovanými klauzulemi). V našem případě jsme dokazovač omezili časovým limitem deseti sekund a paměťový limit jsme dokazovači nechali zvolit automaticky, tedy podle zdrojů dostupných na konkrétním počítači. Další parametry dokazovači říkají, že vstup i výstup (tedy důkaz) mají být ve formátu TPTP, že po skončení požadujeme výpis statistiky o práci dokazovače a že strategie a heuristiky pro dokazování tohoto problému si má dokazovač zvolit automaticky (tedy odhadem podle různých vlastností problému). Zde je výstup dokazovače (pro lepší čitelnost s mírně upraveným zalomením řádků a odsazením):

```
# Problem is unsatisfiable (or provable), constructing proof object
# SZS status Theorem
# SZS output start CNFRefutation.
fof(1,axiom,(
    ! [X1] :
      ( ? [X2] : ( eats(X2,X1) & not_killed_by(X2,X1) )
      => food(X1) ) ),
    file('/john.tptp',eaten_is_food)).

fof(2,axiom,(
    alive(bill) ),
    file('/john.tptp',bill_alive)).

fof(3,axiom,(
    eats(bill,peanuts) ),
    file('/john.tptp',bill_eats_peanuts)).

fof(5,axiom,(
    ! [X2] : ( alive(X2) => ! [X1] : not_killed_by(X2,X1) ) ),
    file('/john.tptp',alive_not_killed_by)).

fof(6,axiom,(
    ! [X1] : ( food(X1) => likes(john,X1) ) ),
    file('/john.tptp',john_likes_food)).
```

[11] http://www.eprover.org/

Pokračování výpisu

```
fof(7,conjecture,(
    likes(john,peanuts) ),
    file('/john.tptp',prove_this)).

fof(8,negated_conjecture,(
    ~ likes(john,peanuts) ),
    inference(assume_negation,[status(cth)],[7])).

fof(9,negated_conjecture,(
    ~ likes(john,peanuts) ),
    inference(fof_simplification,[status(thm)],[8,theory(equality)])).

fof(10,plain,(
    ! [X1] :
      ( ! [X2] : ( ~ eats(X2,X1) | ~ not_killed_by(X2,X1) )
      | food(X1) ) ),
    inference(fof_nnf,[status(thm)],[1])).

fof(11,plain,(
    ! [X3] :
      ( ! [X4] : ( ~ eats(X4,X3) | ~ not_killed_by(X4,X3) )
      | food(X3) ) ),
    inference(variable_rename,[status(thm)],[10])).

fof(12,plain,(
    ! [X3,X4] : ( ~ eats(X4,X3) | ~ not_killed_by(X4,X3) | food(X3) ) ),
    inference(shift_quantors,[status(thm)],[11])).

cnf(13,plain,
    ( food(X1) | ~ not_killed_by(X2,X1) | ~ eats(X2,X1) ),
    inference(split_conjunct,[status(thm)],[12])).

cnf(14,plain,
    ( alive(bill) ),
    inference(split_conjunct,[status(thm)],[2])).

cnf(15,plain,
    ( eats(bill,peanuts) ),
    inference(split_conjunct,[status(thm)],[3])).

fof(19,plain,(
    ! [X2] : ( ~ alive(X2) | ! [X1] : not_killed_by(X2,X1) ) ),
    inference(fof_nnf,[status(thm)],[5])).

fof(20,plain,(
    ! [X3] : ( ~ alive(X3) | ! [X4] : not_killed_by(X3,X4) ) ),
    inference(variable_rename,[status(thm)],[19])).

fof(21,plain,(
    ! [X3,X4] : ( not_killed_by(X3,X4) | ~ alive(X3) ) ),
    inference(shift_quantors,[status(thm)],[20])).

cnf(22,plain,
    ( not_killed_by(X1,X2) | ~ alive(X1) ),
    inference(split_conjunct,[status(thm)],[21])).

fof(23,plain,(
    ! [X1] : ( ~ food(X1) | likes(john,X1) ) ),
    inference(fof_nnf,[status(thm)],[6])).
```

Pokračování výpisu

```
fof(24,plain,(
    ! [X2] : ( ~ food(X2) | likes(john,X2) ) ),
    inference(variable_rename,[status(thm)],[23])).

cnf(25,plain,
    ( likes(john,X1) | ~ food(X1) ),
    inference(split_conjunct,[status(thm)],[24])).

cnf(26,negated_conjecture,
    ( ~ likes(john,peanuts) ),
    inference(split_conjunct,[status(thm)],[9])).

cnf(27,negated_conjecture,
    ( ~ food(peanuts) ),
    inference(spm,[status(thm)],[26,25,theory(equality)])).

cnf(28,plain,
    ( food(X1) | ~ eats(X2,X1) | ~ alive(X2) ),
    inference(spm,[status(thm)],[13,22,theory(equality)])).

cnf(30,plain,
    ( food(peanuts) | ~ alive(bill) ),
    inference(spm,[status(thm)],[28,15,theory(equality)])).

cnf(31,plain,
    ( food(peanuts) | $false ),
    inference(rw,[status(thm)],[30,14,theory(equality)])).

cnf(32,plain,
    ( food(peanuts) ),
    inference(cn,[status(thm)],[31,theory(equality)])).

cnf(33,plain,
    ( $false ),
    inference(sr,[status(thm)],[32,27,theory(equality)])).

cnf(34,plain,( $false ), 33, [proof])

# SZS output end CNFRefutation
# SZS status Theorem
# Initial clauses                      : 7
# Removed in preprocessing             : 0
# Initial clauses in saturation        : 7
# Processed clauses                    : 9
# ...of these trivial                  : 0
# ...subsumed                          : 0
# ...remaining for further processing  : 9
# Other redundant clauses eliminated   : 0
# Clauses deleted for lack of memory   : 0
# Backward-subsumed                    : 0
# Backward-rewritten                   : 0
# Generated clauses                    : 4
# ...of the previous two non-trivial   : 2
# Contextual simplify-reflections      : 0
# Paramodulations                      : 4
# Factorizations                       : 0
# Equation resolutions                 : 0
# Current number of processed clauses  : 9
#    Positive orientable unit clauses  : 2
#    Positive unorientable unit clauses: 0
```

Pokračování výpisu

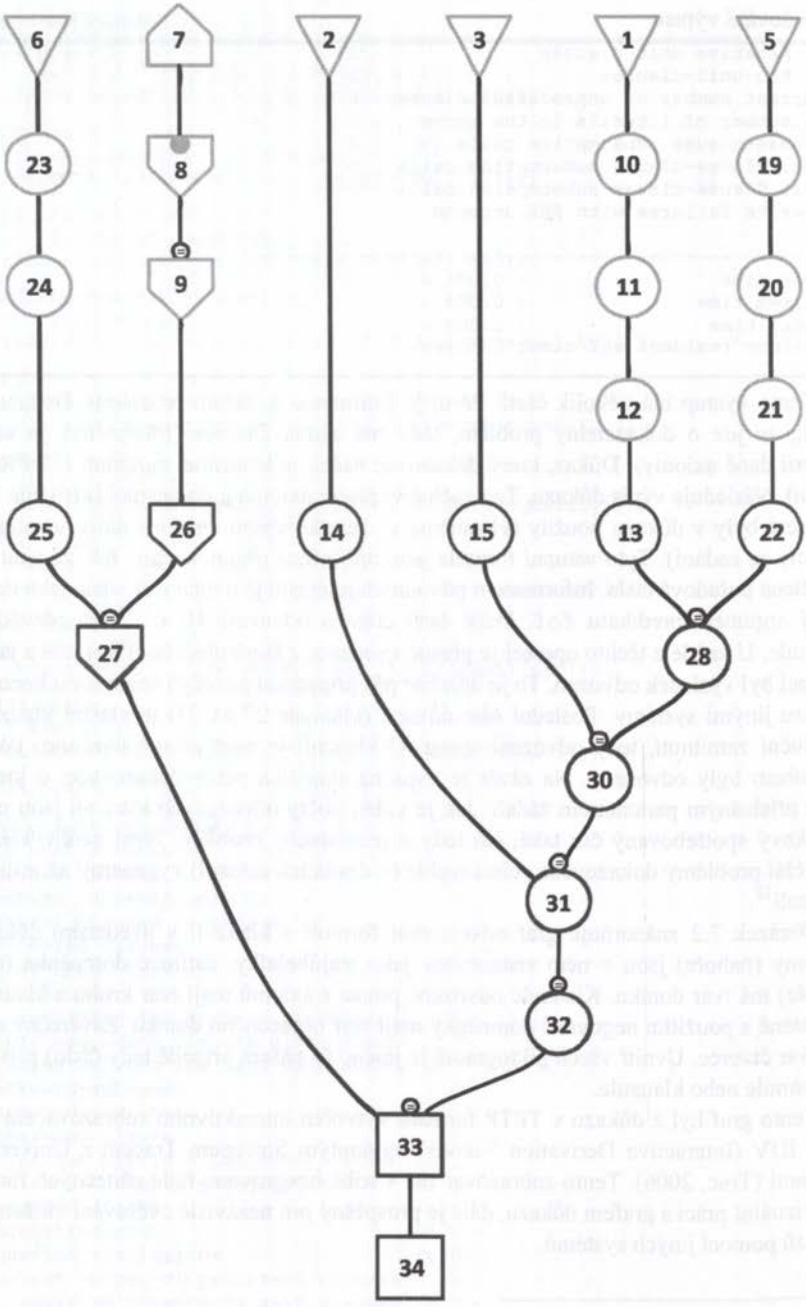
```
#    Negative unit clauses              : 2
#    Non-unit-clauses                   : 5
# Current number of unprocessed clauses: 0
# ...number of literals in the above    : 0
# Clause-clause subsumption calls (NU) : 0
# Rec. Clause-clause subsumption calls : 0
# Unit Clause-clause subsumption calls : 0
# Rewrite failures with RHS unbound    : 0

# ----------------------------------------------
# User time                : 0.004 s
# System time              : 0.004 s
# Total time               : 0.008 s
# Maximum resident set size: 0 pages
```

Tento výstup má několik částí. První je informace o statutu problému. Dokazovač zjistil, že jde o dokazatelný problém, tedy má status *Theorem* (domněnka je větou v teorii dané axiomy). Důkaz, který dokazovač našel, je klauzální zamítnutí (*CNFRefutation*). Následuje výpis důkazu. Ten začíná výpisem axiomů a domněnky (klauzule 1 až 9), které byly v důkazu použity (všimněme si, že zde nejsou uvedeny další nepotřebné axiomy ze zadání). Tyto vstupní formule jsou uniformně přejmenovány tak, že jsou jim přidělena pořadová čísla. Informace o původních jménech je u nich uchována jako dodatečný argument predikátu `fof`. Další částí důkazu (klauzule 10 až 26) je převod na klauzule. U každé z těchto operací je přesně uvedeno, z které předchozí formule a jakou operací byl výsledek odvozen. To je důležité pro případnou pozdější nezávislou kontrolu důkazu jinými systémy. Poslední část důkazu (klauzule 27 až 33) je vlastní klauzální rezoluční zamítnutí, tedy odvození sporu. U klauzulí je opět přesně uvedeno, jakým způsobem byly odvozeny. Na závěr je vypsána statistika práce dokazovače, o kterou jsme příslušným parametrem žádali. Jak je vidět, počty odvozených klauzulí jsou malé a celkový spotřebovaný čas také, šlo tedy o jednoduchý problém. Není neobvyklé, že pro těžší problémy dokazovače velmi rychle (v desítkách sekund) vygenerují až miliony klauzulí[12].

Obrázek 7.2 znázorňuje graf odvozování formulí a klauzulí v uvedeném důkazu. Axiomy (nahoře) jsou v něm znázorněny jako trojúhelníky, zatímco domněnka (také nahoře) má tvar domku. Klauzule odvozené pouze z axiomů mají tvar kruhu a klauzule odvozené s použitím negované domněnky mají tvar obráceného domku. Závěrečný spor má tvar čtverce. Uvnitř všech piktogramů je jméno (v našem případě tedy číslo) příslušné formule nebo klauzule.

Tento graf byl z důkazu v TPTP formátu vytvořen interaktivním zobrazovačem důkazů IDV (Interactive Derivation Viewer) vyvinutým Stevenem Tracem z Univerzity v Miami (Trac, 2006). Tento zobrazovač má v sobě integrovánu řadu užitečných funkcí pro vizuální práci s grafem důkazu, dále je prospěšný pro nezávislé ověřování vložených důkazů pomocí jiných systémů.

[12] To ale neznamená, že důkaz má pak milion kroků. Naprostá většina vygenerovaných klauzulí je obvykle k důkazu nepotřebná.


6
7
2
3
1
5
23
8
10
19
24
9
11
20
12
21
25
26
14
15
13
22
27
28
30
31
32
33
34

Obr. 7.2

7.4.6 Interpretace nalezeného automatické řešení problému

Pokud se převedení slovně zadaných problémů do formálního jazyka vhodného pro automatické dokazování zdá těžké, při pokusech interpretovat automaticky nalezené důkazy obvykle zjistíme, že jde o úlohu ještě těžší. Každý z odvozovacích kroků dokazovače je obvykle snadno pochopitelný, avšak sestavit z nich pokud možno stručné, a snadno pochopitelné slovní vysvětlení je často obtížné. Toto dodatečné lidské chápání je ale důležitá kontrola toho, že jsme problém zformalizovali dobře a že jsme ho automaticky nevyřešili jen díky nějaké formalizační chybě. Námi zvolený logický problém je jednoduchý a jeho řešení také. Důkaz probíhá sporem, tedy:

Předpokládejme, že John nemá rád buráky (8). Protože John má rád všechno jídlo (6, 25), znamenalo by to, že buráky nejsou jídlo (27). Pokud je někdo naživu, tak ho nic nezabilo (5, 22), a pokud někdo něco jí a nezabilo ho to, tak je to jídlo (1, 13). Tedy pokud je někdo naživu a něco jí, tak je to jídlo (28). Bill je naživu (2, 14) a jí buráky (3, 15), tedy buráky jsou jídlo (29, 31, 32). To je spor s (27).

Důkaz sporem je často méně názorný než přímý důkaz, a pokud je to možné, je vhodné důkazy sporem přeformulovat. Zde to jde snadno:

Pokud je někdo naživu, tak ho nic nezabilo (5, 22), a pokud někdo něco jí a nezabilo ho to, tak je to jídlo (1, 13). Tedy pokud je někdo naživu a něco jí, tak je to jídlo (28). Bill je naživu (2, 14) a jí buráky (3, 15), tedy buráky jsou jídlo (29, 31, 32). John má rád všechno jídlo (6, 25), a tedy má také rád buráky.

7.5 Metoda tableau[13]

Další metodou užívanou pro automatické dokazování vět v predikátové logice, ale i v dalších logikách (modálních, temporálních, aj.), je *sémantické tableau*. Sémantické tableau je strom, kde každý uzel je logickou formulí. Sémantické tableau vzniká iterativně z předchozích tableau postupnou aplikací určitých pravidel. Dokazování pomocí metody tableau postupně rozděluje vstupní formuli na menší formule, dokud na všech větvích stromu nenajde komplementární páry formulí, nebo pokud již nemůže aplikovat žádné z pravidel tableau.

Nyní budeme formálně definovat sémantické tableau pro predikátovou logiku. Vstupem pro metodu tableau bude množina formulí predikátové logiky. Cílem metody bude, tak jako v případě rezoluční metody, najít spor v této množině. Takový spor odpovídá existenci komplementárních formulí na všech větvích tableau. Pro jednoduchost budeme předpokládat, že jednotlivé formule budou obsahovat pouze logické spojky: & pro konjunkci, $\vee$ pro disjunkci a $\neg$ pro negaci. Pokud bychom chtěli pracovat i s ostatními logickými spojkami, můžeme je chápat jako zkratky definující uvažované spojky pouze pomocí spojek &, $\vee$ a $\neg$. Tyto zkratky pak před aplikací metody tableau nahradíme.

Na začátku přepokládejme tableau tvořené jen jedním uzlem – kořenem obsahujícím konjunkci všech formulí ze vstupní množiny. Někdy se také tento kořen označuje symbolem T (top). Tento uzel se obvykle při grafickém znázornění tableau jako stromu nezobrazuje.

[13] Tableau [tabló] je francouzské slovo.

Sémantické tableau je tedy strom, který vznikne z kořene postupnou aplikací konečného počtu následujících pravidel. Cílem aplikací těchto pravidel je sestrojit tzv. *uzavřené tableau* – tedy stav, kdy se na každé větvi nachází komplementární pár formulí. Protože jsou následující pravidla logicky korektní, znamená tento stav, že vstupní množina formulí je sporná.

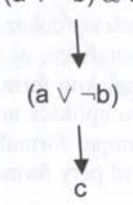
7.5.1 Pravidla tableau

Následující text popisuje jednotlivá pravidla tableau pro predikátovou logiku, která se aplikují na tableau během výpočtu.

Konjunkce (&)
Pokud některá větev v tableau obsahuje formuli tvaru $A \,\&\, B$, kde A a B jsou libovolné formule, připojíme k listu této větve novou větev s formulemi A a B. Formálně toto pravidlo zapíšeme jako

$$(\&)\frac{A \,\&\, B}{\begin{array}{c} A \\ B \end{array}}.$$

Příklad. Obrázek 7.3 ukazuje výsledné tableau po aplikaci pravidla konjunkce na tableau sestávajícího z jediného uzlu $(a \vee \neg b) \,\&\, c$.

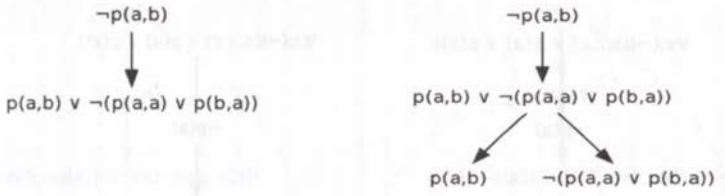


Obr. 7.3

Disjunkce ($\vee$)
Pokud některá větev v tableau obsahuje formuli tvaru $A \vee B$, kde A a B jsou libovolné formule, připojíme k listu této větve dvě nové větve. Jedna větev bude obsahovat formuli A a druhá formuli B. Formálně toto pravidlo zapíšeme takto

$$(\vee)\frac{A \vee B}{A \mid B}.$$

Příklad. Obrázek 7.4 ukazuje počáteční a výsledné tableau po aplikaci pravidla disjunkce.



Obr. 7.4

Negace (¬)

Pokud budeme zpracovávat pomocí metody tableau pouze formule v tzv. *negativní normální formě*[14] (NNF), není třeba žádné pravidlo pro zpracování negace zavádět. Alternativně můžeme zavést těchto pět pravidel:

$$(\neg 1)\frac{\neg\neg A}{A},$$

$$(\neg 2)\frac{\neg(A \,\&\, B)}{\neg A \vee \neg B}, \qquad (\neg 3)\frac{\neg(A \vee B)}{\neg A \,\&\, \neg B},$$

$$(\neg 4)\frac{\neg(\forall x:\varphi)}{\exists x:\neg\varphi}, \qquad (\neg 5)\frac{\neg(\exists x:\varphi)}{\forall x:\neg\varphi}.$$

Univerzální kvantifikace (∀)

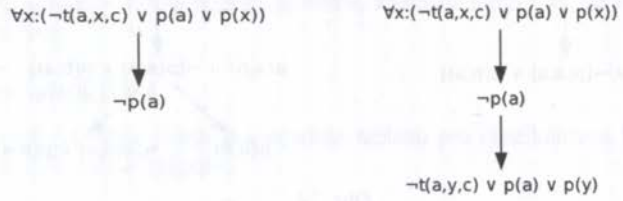
Pokud některá větev v tableau obsahuje formuli tvaru $\forall x:\varphi(x)$, kde x je proměnná a $\varphi(x)$ je formule obsahující volnou proměnnou x, připojíme k listu této větve novou formuli $\varphi(x')$, kde $\varphi(x')$ vznikne z formule $\varphi(x)$ nahrazením všech volných výskytů proměnné x za novou proměnnou x', která se ještě nikde v tableau nevyskytuje, tj.

$$(\forall)\frac{\forall x:\varphi(x)}{\varphi(x')},$$

kde x' je nová proměnná, která se ještě nikde v tableau nevyskytuje.

[14] Formule v negativní normální formě (NNF) je taková logická formule, která obsahuje negaci pouze v literálech. Každou formuli v predikátové logice lze jednouše převést na formuli NNF pomocí De Morganových pravidel (¬(A∨B) ⇔ (¬A & ¬B), ¬(A&B) ⇔ (¬A ∨ ¬B)), pravidel pro propagaci negace přes kvantifikátory (¬(∀x:φ) ⇔ (∃x:¬φ), ¬(∃x:φ) ⇔ (∀x:¬φ)) a pravidla pro rušení vícenásobné negace (¬¬A ⇔ A). Převod všech formulí na formule NNF se obvykle provádí před zahájením metody tableau.

Příklad. Obrázek 7.5 ukazuje počáteční a výsledné tableau po aplikaci pravidla univerzální kvantifikace.



Obr. 7.5

Existenční kvantifikace (∃)

Pokud budeme zpracovávat pomocí metody tableau pouze formule bez existenčních kvantifikátorů (existenční kvantifikátory lze eliminovat tzv. skolemizací) v NNF, není třeba žádné pravidlo pro zpracování existenčního kvantifikátoru zavádět. Alternativně můžeme zavést skolemizační pravidlo:

Pokud některá větev v tableau obsahuje formuli tvaru $\exists x : \delta(x)$, kde x je proměnná a $\delta(x)$ je formule obsahující volnou proměnnou x, připojíme k listu této větve novou formuli $\delta(f(x_1,\ldots,x_n))$, kde $\delta(f(x_1,\ldots,x_n))$ vznikne z $\delta(x)$ nahrazením všech volných výskytů x termem $f(x_1,\ldots,x_n)$. Zde f je nový funkční symbol, který se ještě nikde v tableau nevyskytuje, a $x_1,\ldots,x_n$ jsou všechny volné proměnné formule δ, tj.

$$(\exists)\frac{\exists x : \delta(x)}{\delta(f(x_1,\ldots,x_n))},$$

kde f je nový funkční symbol a $x_1,\ldots,x_n$ jsou volné proměnné formule δ.

Uzavření větve

Pokud se na některé větvi v tableau vyskytují dva literály L a K takové, že existuje unifikační substituce θ formulí L a $\neg K$ (tedy taková nejobecnější substituce, že $L\theta \equiv \neg K\theta$)[15], potom aplikujeme substituci θ na všechny uzly tableau a takovou větev označíme za uzavřenou. Takové větve již není třeba dále rozšiřovat.

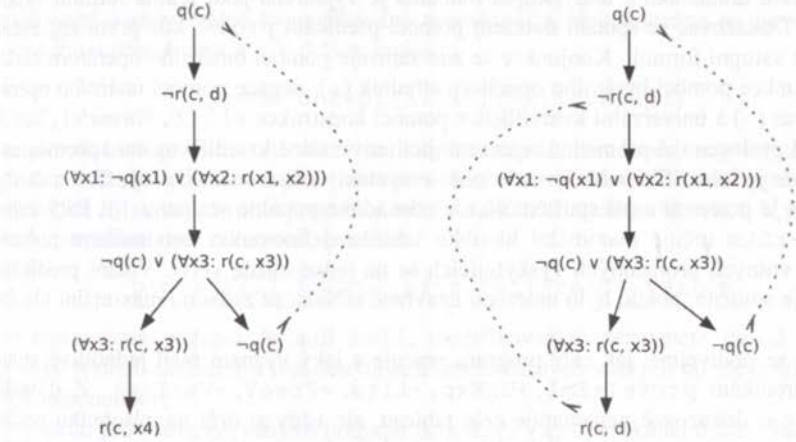
Uzavřené tableau

Řekneme, že tableau je uzavřené, pokud jsou všechny jeho větve uzavřené. Takové tableau dosvědčuje spor vstupní množiny formulí, protože všechna pravidla tableau jsou logicky korektní[16] (tedy kořen tableau implikuje každý uzel v tableau) a každá cesta z kořene do listu obsahuje dvojici komplementárních literálů, což představuje spor.

[15] Například pokud $L = \neg p(X)$ a $K = p(t)$, pak existuje substituce $\theta = \{X/t\}$ taková, že $\neg p(t) \equiv \neg p(t)$.

[16] To co odvozují, lze dokázat ze vstupní množiny formulí.

Příklad. Obrázek 7.6 ukazuje počáteční a výsledné tableau po uzavření levé větve (pravá větev byla již uzavřená). Výsledné tableau je uzavřené.



Obr. 7.6

7.5.2 LeanTAP

V této části si představíme jeden z nejkratších úplných dokazovačů pro predikátovou logiku. Dokazovač se jmenuje *LeanTAP* a jeho autory jsou Bernhard Beckert a Joachim Posegga (Beckert, Posegga, 1994). Celý dokazovač používá metodu tableau a skládá se z pěti klauzulí (pravidel) v programovacím jazyce Prolog. Jeho zdrojový kód je dán předpisem:

```
% prove(+Fml,+UnExp,+Lits,+FreeV,+VarLim)

prove((A,B),UnExp,Lits,FreeV,VarLim) :- !,
   prove(A,[B|UnExp],Lits,FreeV,VarLim).

prove((A;B),UnExp,Lits,FreeV,VarLim) :- !,
   prove(A,UnExp,Lits,FreeV,VarLim),
   prove(B,UnExp,Lits,FreeV,VarLim).

prove(all(X,Fml),UnExp,Lits,FreeV,VarLim) :- !,
   \+ length(FreeV,VarLim),
   copy_term((X,Fml,FreeV),(X1,Fml1,FreeV)),
   append(UnExp,[all(X,Fml)],UnExp1),
   prove(Fml1,UnExp1,Lits,[X1|FreeV],VarLim).

prove(Lit,_,[L|Lits],_,_) :-
   (Lit = -Neg; -Lit = Neg) ->
   ( unify_with_occurs_check(Neg,L)
    ; prove(Lit,[],Lits,_,_)
   ).

prove(Lit,[Next|UnExp],Lits,FreeV,VarLim) :-
   prove(Next,UnExp,[Lit|Lits],FreeV,VarLim).
```

Jako vstup přepokládá dokazovač konjunkci skolemizovaných formulí v NNF.

Pro účely dokazování tedy přepokládáme, že vstupní množina formulí již obsahuje znegovanou domněnku a tato vstupní množina je vyjádřena jako jediná formule (konjunkce). Dokazovač se spouští dotazem pomocí predikátu `prove`, kde první argument obsahuje vstupní formuli. Konjunkce se zde zapisuje pomocí binárního operátoru čárka (`,`), disjunkce pomocí binárního operátoru středník (`;`), negace pomocí unárního operátoru minus (`-`) a univerzální kvantifikátor pomocí konstrukce `all(`*X*`,`*Formule*`)`, kde *X* je nějaká prologovská proměnná reprezentující univerzálně kvantifikovanou proměnnou a *Formule* je kvantifikovaná formule opět v syntaxi dokazovače. Druhý, třetí a čtvrtý argument je pracovní a při spuštění do nich dosadíme prázdné seznamy `[]`. Pátý argument `VarLim` určuje maximální hloubku tableau definovanou maximálním počtem různých volných proměnných vyskytujících se na jedné každé větvi. Volání predikátu `prove` je splněno, pokud bylo nalezeno uzavřené tableau se zadanou maximální hloubkou.

Nyní se podívejme, jak celý program pracuje a jaký význam mají jednotlivé argumenty predikátu `prove(+Fml,+UnExp,+Lits,+FreeV,+VarLim)`. Z důvodů efektivity si dokazovač nepamatuje celé tableau, ale vždy si drží na zásobníku pouze aktuální zpracovávanou větev. Pokud se mu podaří větev úspěšně uzavřít pomocí předposlední klauzule, pokračuje od nejbližšího místa v tableau rozštěpeného pravidlem disjunkce a pokouší se uzavřít další větev. Všimněme si, jak se zde výhodně využívá prologovská unifikace s tzv. occur checkem[17] `unify_with_occurs_check`, která automaticky provede substituci na celé tableau tak, jak to pro uzavření větve v tableau požadujeme. Pokud se dokazovači podaří takto uzavřít všechny větve, končí úspěchem.

Pokud se dokazovač dostane do situace, kdy už má v prvním argumentu jenom literál, ale větev stejně nelze uzavřít (tento případ se zpracovává v poslední klauzuli), uloží si tento literál do seznamu literálů ve třetím argumentu `Lits`[18] a pokračuje dál ve zpracování dosud nezpracovaných formulí ze seznamu v druhém argumentu `UnExp`.

Jednotlivá pravidla tableau se zpracovávají v prvních třech klauzulích. Použití správného pravidla se určí syntakticky podle prvního argumentu. Konjunkce se zpracovává tak, že se predikát `prove` zavolá rekurzivně na první argument konjunkce a druhý argument konjunkce se uloží do seznamu dosud nezpracovaných formulí `UnExp`. Disjunkce se zpracovává tak, že se nejprve rekurzivně zavolá predikát `prove` na první argument disjunkce a potom rekurzivně na druhý argument.

Nyní si vysvětlíme zpracování pravidla univerzální kvantifikace. Protože po aplikaci tohoto pravidla bude do tableau zavedena nová proměnná, je třeba na začátku otestovat, zda jsme již nedosáhli maximálního povoleného počtu různých proměnných. Tento úkol provede volání `\+ length(FreeV,VarLim)`. Ve čtvrtém argumentu `FreeV` si totiž dokazovač ukládá seznam volných proměnných vytvořených během aplikace pravidla pro zpracování univerzálního kvantifikátoru. Následující volání

---

[17] Unifikace s occur checkem je provedení standardní logické unifikace. Prolog z důvodů efektivity neprovádí standardně plnou logickou unifikaci, ale umožňuje unifikovat i například term `f(X)= X`, kde `X` po provedení této unifikace obsahuje nekonečný term `f(f(f(....`, což není logicky korektní. Takovému druhu unifikaci zabraňuje právě vestavěný predikát `unify_with_occurs_check` a jeho kontrola – occur check.

[18] Třetí argument `Lits` tedy obsahuje seznam literálů na právě zpracovávané větvi.

```
copy_term((X,Fml,FreeV),(X1,Fml1,FreeV))
```

vytvoří novou kopii vstupní formule `Fml` s tím, že všechny volné proměnné zůstávají zachované a univerzálně kvantifikovaná proměnná je přejmenována na novou dosud nepoužitou proměnnou `X1`. V dalším kroku

```
append(UnExp,[all(X,Fml)],UnExp1)
```

se původní vstupní klauzule uloží na konec seznamu dosud nezpracovaných formulí `UnExp1`. Tento krok nám zajistí ponechat si možnost vytvářet další volné proměnné pro případ, že by stávající volná proměnná byla v dalších krocích unifikována. V posledním kroku zavoláme opět rekurzivně predikát

```
prove(Fml1,UnExp1,Lits,[X1|FreeV],VarLim)
```

se zpracovanou vstupní formulí `Fml1`, modifikovaným seznamem dosud nenavštívených formulí `UnExp1` a rozšířeným seznamem různých volných proměnných `FreeV` o proměnnou `X1`.

Pokud pro zvolenou vstupní hodnotu parametru `VarLim` skončí dokazovač neúspěchem, znamená to, že neexistuje uzavřené tableau, v němž je v každé větvi pravidlo univerzální kvantifikace použito nejvýše `VarLim`-krát. Abychom zajistili úplnost popsané dokazovací procedury, je třeba po takovém neúspěchu postupně zvyšovat hodnotu parametru `VarLim`. Odtud je také zřejmé, že tento postup, v případě neexistence důkazu, nemusí pro žádnou konkrétní hodnotu parametru `VarLim` úspěšně skončit.

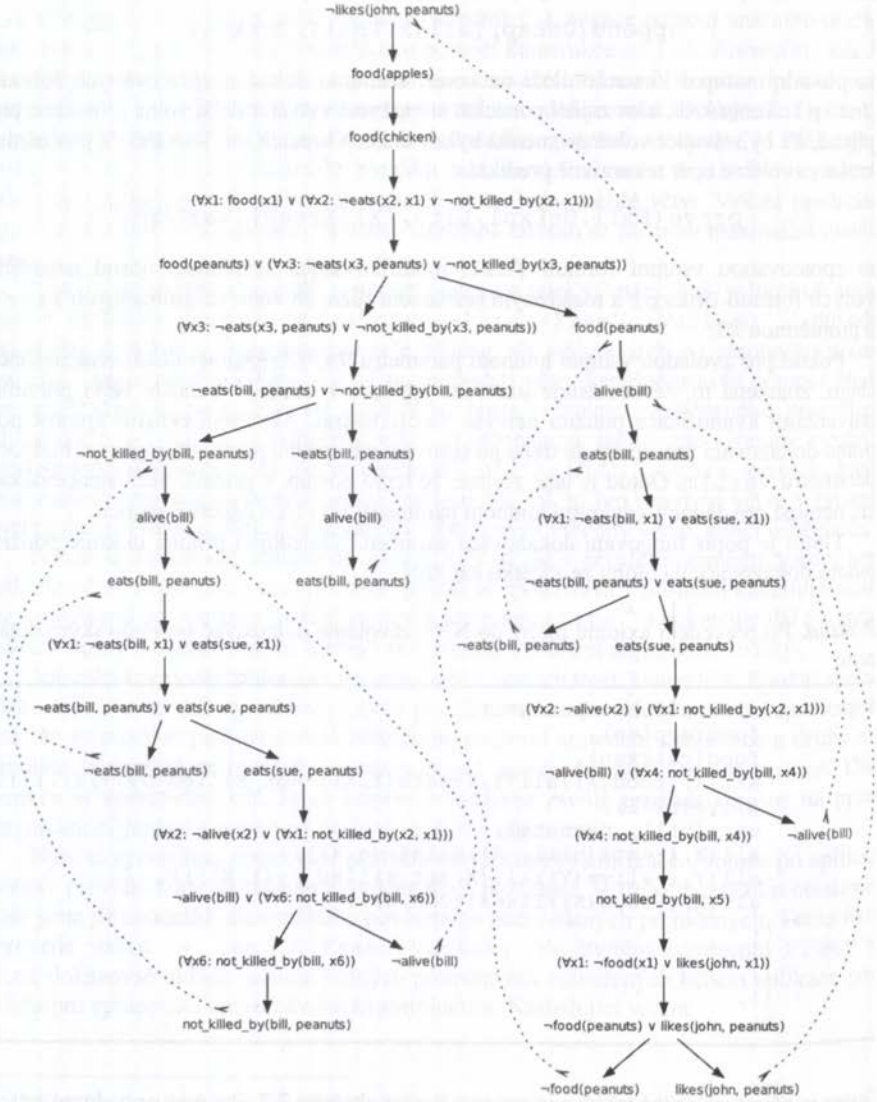
Tímto je popis fungování dokazovače ukončen. Následující příklad ukazuje použití tohoto dokazovače na úlohu ze začátku kapitoly.

Příklad. Po převedení axiomů úlohy do NNF zavoláme dokazovač prologovským dotazem:

```
?-prove((-likes(john, peanuts),
         food(apples),
         food(chicken),
         all(X,(food(X);all(Y,(-eats(Y,X);-not_killed_by(Y,X))))),
         alive(bill),
         eats(bill, peanuts),
         all(X,(-eats(bill, X);eats(sue,X))),
         all(Y,(-alive(Y);all(X,not_killed_by(Y,X)))),
         all(X,(-food(X);likes(john,X)))
        ),
        [],
        [],
        [],
        5).
```

Dotaz uspěje a nalezené tableau je znázorněno na obrázku 7.7. Pokuste se o slovní interpretaci tohoto důkazu sporem, podobně jak jsme to učinili pro rezoluční důkaz v odst. 7.4.6.

I po více než 15 letech je myšlenka LeanTAP, tedy dokazovače tableau v Prologu, stále živá, jak dokládají v současnosti se stále zdokonalující dokazovače LeanCoP (Otten, 2008) a MaLeCoP (Urban, 2011).



Obr. 7.7

7.6 Kam dále?

Tato kapitola si kladla za cíl stručně představit obor automatického uvažování a jeho základní pojmy a metody. Našim cílem bylo také seznámit čtenáře se současnými i potenciálními aplikacemi a předvést praktické použití automatických dokazovačů. Řada metod a aplikací mohla být ve vymezeném rozsahu uvedena pouze krátce, přestože jde často o rychle se rozvíjející vědní disciplíny používané v mnohamilionových projektech. Na závěr proto uveďme několik dalších odkazů pro čtenáře s hlubším zájmem o teorii a použití automatického uvažování.

Vyčerpávající moderní příručkou automatického uvažování je dvousvazková *Handbook of Automated Reasoning* (Robinson, Voronkov, 2001). Na praxi orientovaný moderní průvodce s řadou implementovaných algoritmů je *Handbook of Practical Logic and Automated Reasoning* (Harrison, 2009). Starší, volněji pojatá a volně elektronický dostupná je kniha *Computational Modelling of Mathematical Reasoning* (Bundy, 1983).

Literatura

Andrews P. B., Brown C. E.: TPS: A Hybrid Automatic-Interactive System for Developing Proofs, *Journal of Applied Logic*, 4 (2006), s. 367–395.

Beckert B., Posegga J.: LeanTAP: Lean Tableau-based Deduction. *12th International Conference on Automated Deduction* (*CADE*), Springer 1994, s. 793–797.

Benzmüller C., Paulson L. C., Theiss F., Fietzke A.: LEO-II – A Cooperative Automatic Theorem Prover for Classical Higher-Order Logic (System Description). *IJCAR 2008*, s. 162–170.

Bertot Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. In: *Theoretical Computer Science*. Springer, 2004.

Brown C. E.: Satallax: An Automatic Higher-Order Prover. *IJCAR 2012*, s. 111–117.

Bundy A.: *The Computer Modelling of Mathematical Reasoning*. Academic Press, Inc., 1983.

Claessen K., Sörensson N.: New techniques that improve MACE-style finite model finding. In: *Proceedings of the CADE-19 Workshop on Model Computation*, 2003.

Dahn B. I.: Robbins algebras are Boolean: A revision of McCune's computer-generated solution of Robbins problem, *Journal of Algebra*, 208, 1998, s. 526–532.

Eén N., Sörensson N.: An Extensible SAT-solver. *SAT*, 2003, s. 502–518.

Hillenbrand T., Buch A., Vogt R., Löchner B.: WALDMEISTER – High-Performance Equational Deduction, *J. Autom. Reasoning* 18(2), 1997, s. 265–270.

Harrison J.: Floating-Point Verification, *J. UCS* 13(5), 2007, s. 629–638.

Harrison J.: *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

Harrison J.: HOL Light: An Overview. *TPHOLs 2009*, s. 60–66.

Kaufmann M., Moore J. S.: How Can I Do That with ACL2? Recent Enhancements to ACL2. *ACL2 2011*, s. 46–60.

Korovin K.: iProver – An Instantiation-Based Theorem Prover for First-Order Logic (System Description). *IJCAR 2008*, s. 292–298.

Mahajan Y. S., Zhaohui Fu, Malik S.: Zchaff2004: An Efficient SAT Solver. *SAT* (Selected Papers) 2004, s. 360–375.

McCune W. W.: *Otter 3.3 Reference Manual*. Technical Report ANL/MSC-TM-263, Argonne National Laboratory, Argonne, USA, 2003.

De Moura L., Bjørner N.: Z3: An Efficient SMT Solver. *TACAS 2008*, s. 337–340.

Nieuwenhuis R., Oliveras A.: Decision Procedures for SAT, SAT Modulo Theories and Beyond. The BarcelogicTools. *LPAR 2005*, s. 23–46.

Nipkow T., Paulson L. C., Wenzel M.: *Isabelle/HOL – A Proof Assistant for Higher-Order Logic*. Springer, 2002.

Nonnengart A., Weidenbach Ch.: Computing Small Clause Normal Forms, *Handbook of Automated Reasoning 2001*, s. 335–367.

Otten J.: leanCoP 2.0 and ileanCoP 1.2: High Performance Lean Theorem Proving in Classical and Intuitionistic Logic (System Descriptions). In: *Proceedings of the 4th international joint conference on Automated Reasoning (IJCAR '08)*. Springer 2008, s. 283–291.

Owre S., Shankar N.: A Brief Overview of PVS. *TPHOLs 2008*, s. 22–27.

Phillips J. D., Stanovský D.: Automated theorem proving in quasigroup and loop theory, *AI Commu*, 23(2–3), 2010, s. 267–283.

Riazanov A., Voronkov A.: The design and implementation of VAMPIRE, *AI Commun*, 15(2–3), 2002, s. 91–110.

Robinson J. A., Voronkov A. (eds): *Handbook of Automated Reasoning*. 2 volumes, Elsevier and MIT Press, 2001, ISBN 0-444-50813-9, 0-262-18223-8.

Schulz S.: E – a brainiac theorem prover, *AI Commun*, 15(2–3), 2002, s. 111–126.

Sutcliffe G., Suttner C. B.: The state of CASC, *AI Commun*, 19(1), 2006, s. 35–48.

Sutcliffe G., Suttner C. B.: The TPTP Problem Library – CNF Release v1.2.1, *J. Autom. Reasoning* 21(2), 1998, s. 177–203.

Sutcliffe G.: *Webová stránka knihovny a jazyka TPTP* http://www.tptp.org, *webová stránka pro spouštění systémů na TPTP úlohách*: http://www.tptp.org/cgi-bin/SystemOnTPTP.

Trac S., Puzis Y., Sutcliffe G.: An Interactive Derivation Viewer, *Electr. Notes Theor. Comput. Sci.* 174(2), 2007, s. 109–123.

Urban J., Sutcliffe G.: ATP Cross-Verification of the Mizar MPTP Challenge Problems. *LPAR 2007*, s. 546–560.

Urban J., Vyskočil J., Štěpánek P.: MaLeCoP: machine learning connection prover. In: *Proceedings of the 20th international conference on Automated reasoning with analytic tableaux and related methods (TABLEAUX'11)*. Springer, 2011, s. 263–277.

Urban J.: MPTP 0.2: Design, Implementation, and Initial Experiments, *J. Autom. Reasoning*, 37(1–2), 2006, s. 21–43.

Urban J.: MoMM – Fast Interreduction and Retrieval in Large Libraries of Formalized Mathematics, *International Journal on Artificial Intelligence Tools*, 15(1), 2006, s. 109–130.

Weidenbach C., Dimova D., Fietzke A., Kumar R., Suda M., Wischnewski P.: SPASS Version 3.5. In: Schmidt R. A. (ed.): *Proceedings of the 22nd International Conference on Automated Deduction (CADE-22)*. Springer, 2009, s. 140–145.

8 Modální a temporální logiky

Sémantika výrokové a predikátové logiky některým svým tvrzením (formulím) přiřazuje pravdivostní hodnoty například „formule A je pravdivá" nebo „formule B je nepravdivá". Taková konstatování se vždy vztahují k určité konkrétní interpretaci jazyka. Je ovšem celkem přirozené předpokládat, že ke tvrzením A či B lze přiřadit i další konstatování, například taková, která vyjadřují *modality* typu „je nutné, aby platilo tvrzení A" nebo „je možné, aby platilo tvrzení B".

Taková konstatování mají praktický význam, mohou vyjadřovat uskutečnitelnost určitých kroků, pravděpodobnost toho, že nějaká okolnost nastane, či další vyjádření o stavu reálného světa, která nelze pomocí přesné, ale úzce pojaté pravdivosti vyjádřit.

To si uvědomovali již antičtí filosofové a výsledky jejich zkoumání o modalitách shrnul Aristotelés ve svých spisech. Moderní historie modální logiky začíná ve dvacátém století. C. I. Lewis (1918) nebyl spokojen s obvyklou (materiální) implikací $A \rightarrow B$, která se běžně používá pro vyjádření příčiny a následku, například říkáme-li: „Když je káva sladká, moc mi chutná." Ovšem s toutéž spojkou implikace se setkáme i ve větě: „Když je káva sladká a má příchuť benzinu, moc mi chutná.", kterou bychom v běžném životě těžko považovali za pravdivou. Přesto ve výrokové logice platí, že nezávisle na pravdivostním ohodnocení první věty, je druhá věta určitě pravdivá. To je důsledkem základní vlastnosti materiální implikace, která je definována tak, že z nepravdivého předpokladu plyne cokoliv. Jinými slovy řečeno: Je-li předpoklad A při uvažovaném ohodnocení nepravdivý, pak je formule $A \rightarrow B$ pravdivá nezávisle na tom, o čem B hovoří. Tak se může stát, že pravdivé ohodnocení dostanou i formule, které nám mohou připadat absurdní, neboť dávají do souvislosti údaje zcela neslučitelné. C. I. Lewis proto definoval takzvanou *striktní implikaci*, která klade přísnější požadavky na vztah mezi významem svých předpokladů a závěrů. Použil k tomu operátor $\Diamond$ „je možné", negaci $\neg$ a konjunkci $\&$. Striktní implikaci označenou $A \prec B$ definoval pomocí formule

$$A \prec B, \text{ právě když } \neg\Diamond(A \,\&\, \neg B).$$

Povšimněme si, že striktní implikace („káva je sladká" $\prec$ „moc mi chutná") neplatí, a to právě kvůli tomu, že je možné si představit situaci, kdy platí („káva je sladká"$\&\neg$„moc mi chutná"). Ze stejného důvodu neplatí ani striktní implikace („káva je sladká"$\&$ „káva má příchuť benzínu" $\prec$ „moc mi chutná").

Dnes patří striktní implikace spíše k historickým kuriozitám, ale operátor možnosti $\Diamond$, popisující modalitu „je možné", byl na světě. Není to logická spojka, výraz $\Diamond C$ není formulí ani výrokové, ani predikátové logiky, ale formulí modální logiky, ve které platnost modální formule $\Diamond C$ nezáleží jen na tvaru formule C, ale také na širším kontextu. Tuto skutečnost zachycuje *sémantika možných světů*, která umožňuje uvažovat nejen o světě, v němž se právě nacházíme, ale i o světech, jež považujeme za možné. To je zásadní rozdíl oproti výrokové logice, v níž je možné vyčíst pravdivostní hodnotu formule A pouze ze syntaxe formule A skládáním pravdivostních hodnot jejích komponent pomocí pravdivostních tabulek logických spojek. Původní Lewisův systém **S** odpovídá v dnešní terminologii, které bude věnována tato kapitola, modální logice S3, častěji jsou ale používané silnější modální logiky S4 a S5.

8.1 Úvod do modální logiky

Výklad modální logiky dnes vychází z operátoru nutnosti „je nutné", který je duální k operátoru možnosti $\Diamond$ a používá se pro něj znak $\Box$. Je-li A nějaká formule modální logiky, formuli $\Box A$ čteme „je nutné A". Modální logika se původně soustředila na studium nutnosti z hlediska jediného subjektu. Ale později se ukázalo, že jazyk modální logiky lze přirozeně rozšířit tak, aby bylo možné současně brát v úvahu konečný počet agentů.

Právě takto zobecněná modální logika má mnoho aplikací, které výklad slova „nutnost" přizpůsobují oblasti svého použití. Záleží na tom, zda ji chceme používat pro formulaci logické hádanky, pro popis znalostí hráčů nějaké karetní hry či účastníků obchodního jednání. Setkáváme se s ní v průmyslu, při analýze složitých situací vznikajících v multiagentních či distribuovaných systémech, ve filosofii i jinde. Možnosti modální logiky se nejčastěji ukazují prostřednictvím formalizace úvah, které se běžně provádějí v informatice při analýze systémů zasílajících si zprávy nebo při popisu práce různých automatů.

Nejprve uvedeme několik jednoduchých příkladů ze života, abychom přiblížili použití modální logiky ve světě znalostí. Víme, že nositeli znalostí mohou být osoby, paměťová média, procesory, znalostní báze nebo prvky distribuovaného systému. Pro všechny nositele znalostí budeme používat společný název agenti. Z hlediska znalosti je podstatné to, zda agent nějakou informaci ví či neví. Abychom zdůraznili, že v dalším výkladu se hodláme soustředit právě na způsob práce s informacemi, které agent ví, nahradíme neutrální znak modality $\Box$ písmenem K (knowledge), k němuž připojíme index nositele. Formule $K_i\alpha$ vytvořená využitím modality K_i vyjadřuje skutečnost „agent i ví, že tvrzení α platí" (jinými slovy tvrzení α je z hlediska agenta i nutně platné). Použijeme-li například zkratky p a q pro následující dvě tvrzení:

„v posteli leží babička",
„Karkulka je snědená",

můžeme pomocí modalit s indexy Karkulka, vlk a myslivec popsat řadu scén z pohádky o Červené Karkulce. Třeba situaci *„Karkulka neví, že v posteli není babička, vlk to ale ví"* popisuje formule

$$\neg K_{\text{Karkulka}} \neg p \;\&\; K_{\text{vlk}} \neg p.$$

Podobně můžeme postupovat i ke složitějším situacím – třeba informaci *„Vlk ví, že Karkulka neví, že v posteli není babička, a Karkulka neví, že vlk ví, že v posteli není babička"* přesně popisuje formule

$$K_{\text{vlk}} \neg K_{\text{Karkulka}} \neg p \;\&\; \neg K_{\text{Karkulka}} K_{\text{vlk}} \neg p.$$

Můžeme také popsat informaci, že *myslivec neví, jestli je Karkulka snědená*? Jinými slovy chceme vyjádřit, že *„Myslivec neví (není si jist) ani tím, že Karkulka je snědená, ani tím, že Karkulka není snědená."* Tomu tedy odpovídá formule

$$\neg K_{\text{myslivec}} q \;\&\; \neg K_{\text{myslivec}} \neg q,$$

kterou lze podobně jako ve výrokové logice zapsat ekvivalentně také pomocí disjunkce ve tvaru

$$\neg(K_{\text{myslivec}} q \vee K_{\text{myslivec}} \neg q).$$

Můžeme zapsat i daleko složitější situace. Například takovou, ve které platí, že *myslivec neví, jestli vlk ví, že myslivec neví, že vlk ví, že Karkulka je snědená*

$$\neg K_{\text{myslivec}}(K_{\text{vlk}} \neg K_{\text{myslivec}} K_{\text{vlk}} q) \;\&\; \neg K_{\text{myslivec}} \neg (K_{\text{vlk}} \neg K_{\text{myslivec}} K_{\text{vlk}} q).$$

V podobně složitých případech použití modalit znalostí v hovorovém jazyce se většina lidí rychle ztrácí, jak dobře demonstruje příklad z aféry Watergate (1971) převzatý ze základní publikace věnované logické analýze znalostí pro více jednajících subjektů (Fagin, 2003): *Dean neví, jestli Nixon ví, že Dean ví, že Nixon ví, že se McCord vloupal do O'Brienovy kanceláře ve Watergate.*

Uvedné tvrzení je jistě důležité pro to, jak má Dean před senátním výborem vypovídat. Označíme-li Deana za agenta 1, Nixona za agenta 2 a r za výrok „*McCord se vloupal do kanceláře O'Briena ve Watergate*". Pak můžeme uvedené tvrzení zapsat takto

$$\neg K_1(K_2K_1K_2r) \;\&\; \neg K_1\neg(K_2K_1K_2r).$$

Za povšimnutí jistě stojí, že tato formule má velmi podobnou strukturu jako naše poslední formule o myslivci a vlkovi. Srovnání toho, jak dobře rozumíme obsahu obou těchto formulí, potvrzuje, jak náročné je pro člověka v témže okamžiku uvažovat o světě již z pohledu dvou subjektů. Kdykoliv pracujeme se znalostmi více subjektů, hraje významnou úlohu svět, ve kterém se agenti pohybují, například *les*, *senátní výbor* a *protokol z dosavadních výpovědí*. Ovšem naše porozumění vytvářeným formulím významně závisí i na tom, jak je nám tento svět blízký – jak dobře se v něm orientujeme. Není pochyb, že by bylo velmi užitečné mít k dispozici formální aparát, který by pomohl odvozovat důsledky formulí, které obsahují modální operátory. Právě to si klade za cíl studium modální logiky.

8.1.1 Sémantika modální logiky

I když budeme pokračovat ve výkladu modální logiky znalostí, upozorňujeme, že úvahy podobné těm uváděným pro modální operátor K_i mohou být provedeny i pro jiné modální operátory s poněkud odlišnou interpretací, například pro operátor B_i, který popisuje to, čemu agent i věří.

Zatím jsme pouze zavedli modality K_i jako nové prvky pro tvorbu formulí. Vzniklým formulím je však třeba dát i náležitou interpretaci. Budeme postupovat podobně jako v případě výrokové logiky, kde interpretaci vytváříme prostřednictvím ohodnocení prvotních formulí a tabulek pro pravdivostní ohodnocení booleovských spojek. Dřív, než přistoupíme k formálnímu zavedení pojmu interpretace jazyka modální logiky, zamysleme se nad nástroji, které běžně používáme, pokud uvažujeme o znalostech více subjektů. Jako velmi pěkný příklad na tomto místě poslouží klasická hádanka o králi jeho třech rádcích.

Příklad 8.1. Král se rozhodl, že si ověří, jestli jsou jeho tři rádci opravdu tak chytří, jak se staví. Do prázdně truhly dal 5 klobouků navlas stejného tvaru, z nichž 3 byly bílé a 2 černé. Pomocí těchto rekvizit připravil pro své rádce malé intelektuální cvičení. Pozval všechny tři společně do své komnaty, kam museli postupně vejít úplně zatemněnou předsíní, kde každému z nich někdo ze sloužících dal na hlavu jeden z klobouků z truhly s tím, že klobouk si v žádném případě nesmí sundat. Samozřejmě, že rádce v temné předsíni nemohl rozpoznat barvu svého klobouku a v královské komnatě zalité sluncem nebyla žádná zrcadla, která by mu barvu prozradila.

Všichni tři rádci byli pak opět současně usazeni kolem kulatého stolu a král jim vysvětlil, o co mu jde. Popsal jim přesně obsah truhly se 3 bílými a 2 černými klobouky a zeptal se jich: „Milí rádci, jsem zvědav, zda přijdete pouhou silou svého rozumu na to, jaký má kdo z Vás klobouk. Odpovídejte v pořadí, jak jste vešli do místnosti." Rádci se obezřetně rozhlédli kolem sebe a postupně promlouvali takto:

Odpověď **O1** Rádce1 (ten, který vešel první): Nevím, jaký mám klobouk.
Odpověď **O2** Rádce2 (ten, který vešel druhý): Ani já nevím, jaký mám klobouk.
Odpověď **O3** Rádce3 (ten, který vešel poslední): Já už vím, jaký mám klobouk.

Podaří se i nám uhodnout, jaké barvy klobouků dostali jednotliví rádci? Samozřejmě musíme vycházet z toho, že jakékoli přidělení klobouků je možné. Výčet všech přípustných alternativ je na obr. 8.1a), který znázorňuje všech 2^3 možných situací a současně také upozorňuje na jedinou, kterou je třeba předem vyloučit, neboť odporuje zadání úlohy. Je to situace orámovaná čárkovaně a označená „krok 0", v níž všichni tři rádci mají černý klobouk – tato situace je určitě vyloučená, neboť v truhle byly jen 2 černé klobouky.

Kdyby Rádce1 viděl před sebou 2 černé klobouky, muselo by mu být jasné, že na něj jiný než bílý nezbyl a určitě by neodpověděl podle **O1**. Tedy díky odpovědi **O1** můžeme vyloučit další alternativu tak, jak je zvýrazněno na obr. 8.1b) – orámování situace vyloučené díky právě provedené úvaze je opět změněno ze silného na čárkované a pod odpovídající skupinou je poznámka „krok 1".

Rádce2 musí řešit daleko složitější situaci, protože musí brát v úvahu nejen okolnosti zadání, ale i odpověď **O1** svého předchůdce. Určitě tedy ví, že obr. 8.1b) shrnuje všechny zatím přípustné alternativy. Říká-li pak **O2**, pak je zřejmé, že ani Rádce2 nevidí 2 černé klobouky a počet možných alternativ se zmenšuje na pouhých 5 možností tak, jak naznačuje obr. 8.1c) a „krok2a". Opravdu je všech pět zbylých možností v souladu s odpovědí **O2**?

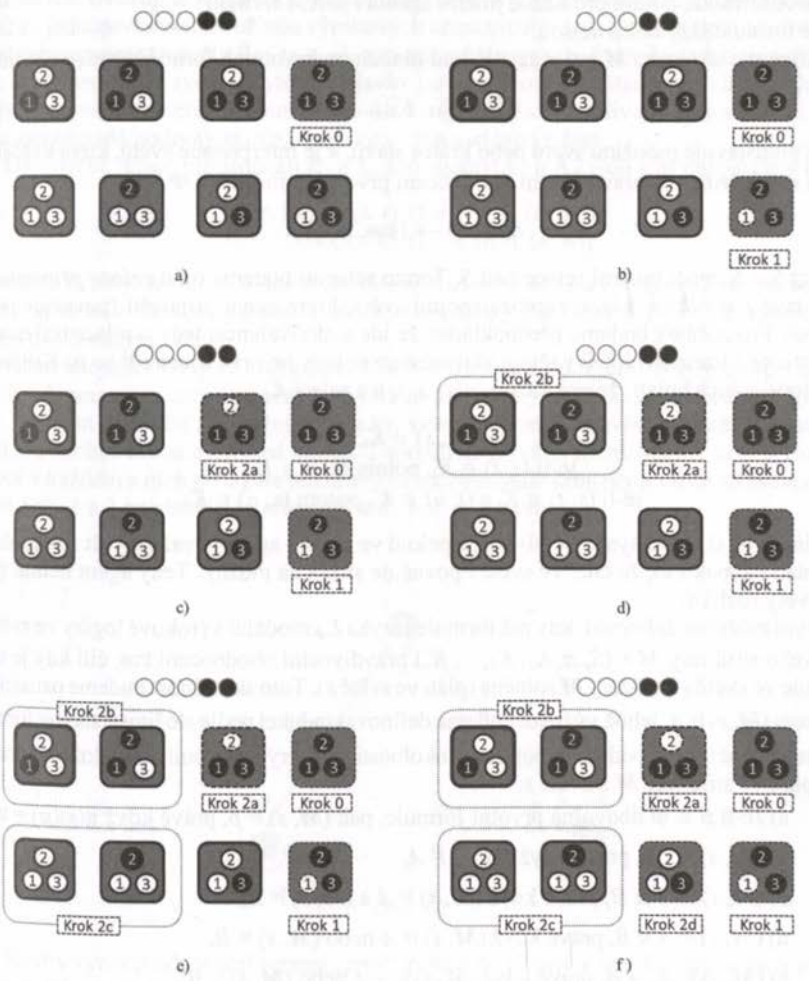
Předpokládejme, že Rádce2 má buď bílý, nebo černý klobouk a při tom vidí Rádce1 v černém a Rádce3 v bílém klobouku. Jde o dvě situace, které zatím nedokáže Rádce2 od sebe odlišit, což je naznačeno v obr. 8.1d) jako „krok2b". Každopádně v této situaci musí říci **O2**. Podobně je tomu i v případě, že Rádce2 vidí oba své partnery v bílých kloboucích, obr. 8.1e) a „krok2c". I v této situaci musí Rádce2 říci **O2**.

Zbývá zvážit, jak by měl Rádce2 odpovědět, pokud by nastala poslední z přípustných 5 možností, kdy vidí Rádce1 v bílém a Rádce3 v černém klobouku. Za těchto okolností je zřejmé, že Rádce2 nemůže mít jiný než bílý klobouk a jeho odpověď by jistě nemohla být **O2**. Proto je tato možnost vyloučena tak, jak ukazuje „krok2d" na obr. 8.1f).

Všechny právě provedené úvahy docela jistě použil i Rádce3, který tedy ví, že v souladu s dosud získanými informacemi jsou pouze 4 silně orámované alternativy na obr. 8.1f), které zbyly z původních 8 uvedených na obr. 8.1a). A ve všech z nich má bílý klobouk. Může tedy radostně ohlásit **O3**. Je zajímavé, že první dva rádci zatím nejsou o nic blíž pravdě než na začátku.

Uvedený příklad demonstruje postup, který člověk používá, pokud uvažuje o vnějším světě, o němž nemá úplné informace. Takové okolnosti člověka nutí, aby zvažoval různé alternativy – možné světy. Právě na práci s možnými světy je založena Kripkeho sémantika výrokové modální logiky, které se také říká sémantika možných světů. V Kripkeho struktuře jsou všechny světy popsány stejným jazykem. K popisu vnějšího světa volíme jen nejnutnější fakta, která jsou důležitá pro popis a řešení dané situace, např. „*babička je v posteli*", „*je hezky*", „*McCord se vloupal do kanceláře O'Briena ve Watergate*" atd.

Z hlediska logiky jde o elementární (definitivní) tvrzení, která nepotřebují dalšího vysvětlení. Jsou to vlastně výrokové konstanty, kterým se také říká prvotní formule. K popisu světa a případných akcí takové nástroje plně postačí, i když hodláme budovat jazyk obecnější, totiž jazyk výrokové modální logiky, ve kterém vedle výrokových spojek budou i nové operátory označující modality. Postup konstrukce složených formulí tohoto jazyka bude velmi podobný tomu, který je použit například v kapitolách (Štěpánková, Štěpánek, 1993) nebo (Urban a spol., 2012).



Obr. 8.1

Pro každou úlohu budeme množinu prvotních formulí označovat symbolem Φ a z praktických důvodů budeme předpokládat, že to je konečná množina. Tento předpoklad není nutný, ale je žádoucí, chceme-li úlohu zpracovávat počítačem. Totéž platí i o dalších komponentách Kripkeho struktur, s jejichž pomocí budeme definovat sémantiku formulí výrokové modální logiky. Zvolíme-li množinu Φ dostatečně velikou, získáme popis světů s jemnou granularitou, která postačí k řešení velkého počtu úloh.

Volbou množiny prvotních formulí Φ určíme jazyk L_Φ výrokové modální logiky, jehož formule se definují indukcí takto:

- Každá prvotní formule $p \in \Phi$ je formulí.
- Jsou-li A, B formule, potom $\neg A$ je formulí a $A \,\&\, B$ je formulí.

– Je-li A formule, potom pro každé jméno agenta i je K_iA formulí.
– Jiné formule jazyka L_Φ nejsou.

Kripkeho struktura $\boldsymbol{M}$ pro n agentů nad množinou prvotních formulí Φ je $(n+2)$-tice

$$(S, \pi, K_1, K_2, \ldots, K_n),$$

kde S představuje množinu světů nebo krátce stavů, π je interpretace světů, která každému světu $s \in S$ přiřazuje pravdivostní ohodnocení prvotních formulí z Φ, tedy

$$\pi(s) : \Phi \rightarrow \{\text{true}, \text{false}\},$$

a $K_i \subseteq S \times S$ značí binární relace nad S. Těmto relacím budeme říkat *relace přípustnosti* nebo také *posibilitní relace*, protože spojují světy, které agent připouští (považuje je za možné). Pro začátek budeme předpokládat, že jde o ekvivalence, tedy o relace reflexivní, symetrické a tranzitivní. Díky těmto vlastnostem se tedy nejprve soustředíme na Kripkeho struktury, v nichž platí, že pro každé světy s, t, u a relaci K_i

$$(s, s) \in K_i,$$
$$\text{je-li } (s, t) \in K_i, \text{ potom } (t, s) \in K_i,$$
$$\text{je-li } (s, t) \in K_i \text{ a } (t, u) \in K_i, \text{ potom } (s, u) \in K_i.$$

Povšimněme si, že ze symetrie plyne, že pokud ve světě s agent považuje svět t za možný, pak platí naopak i to, že také ve světě t považuje svět s za možný. Tedy agent neumí tyto dva světy rozlišit.

Nyní můžeme definovat, kdy má formule jazyka L_Φ modální výrokové logiky ve světě s Kripkeho struktury $\boldsymbol{M} = (S, \pi, K_1, K_2, \ldots, K_n)$ pravdivostní ohodnocení true, čili kdy je tato formule ve světě s struktury $\boldsymbol{M}$ splněna (platí ve světě s). Tuto skutečnost budeme označovat výrazem $(\boldsymbol{M}, s) \models A$, jehož význam budeme definovat indukcí podle složitosti stavby formulí. K tomu stačí, když podrobně popíšeme okolnosti, za kterých prvotní nebo složená formule A platí ve struktuře $\boldsymbol{M}$ a světě s:

a) Je-li $p \in \Phi$ libovolná prvotní formule, pak $(\boldsymbol{M}, s) \models p$, právě když $\pi(s)(p) = \text{true}$,

b) $(\boldsymbol{M}, s) \models \neg A$, právě když $(\boldsymbol{M}, s) \not\models A$,

c) $(\boldsymbol{M}, s) \models A \,\&\, B$, právě když $(\boldsymbol{M}, s) \models A$ a $(\boldsymbol{M}, s) \models B$,

d) $(\boldsymbol{M}, s) \models A \vee B$, právě když $(\boldsymbol{M}, s) \models A$ nebo $(\boldsymbol{M}, s) \models B$,

e) $(\boldsymbol{M}, s) \models A \rightarrow B$, právě když $(\boldsymbol{M}, s) \models \neg A$ nebo $(\boldsymbol{M}, s) \models B$,

f) $(\boldsymbol{M}, s) \models K_iA$, právě když $(\boldsymbol{M}, t) \models A$ pro každé t, $(s, t) \in K_i$.

Stojí za povšimnutí, že platnost formule K_iA ve světě s je definována prostřednictvím platnosti formule A v množině dalších světů – přesněji ve všech světech t, které agent i považuje ve světě s za možné. Tedy platnost formule K_iA ve světě s se odvozuje z platnosti (jiné jednodušší) formule A ve všech světech t takových, že $(s, t) \in K_i$. To je také důvod, proč hovoříme o modalitě K_i jako o operátoru – nelze jej totiž chápat jako výrokovou spojku, jejíž platnost lze jednoznačně charakterizovat pomocí booleovské tabulky, která vždy kombinuje pouze údaje vztahující se k jedinému světu, jako je tomu v případě pravidel a) – e).

Kripkeho struktura se velmi přirozeně dá znázornit ve formě grafu, jehož uzly odpovídají množině uvažovaných světů a hrany relacím přípustnosti jednotlivých agentů. Uveďme pro ilustraci velmi jednoduchý příklad převzatý z monografie (Fagin a kol., 1995 a 2003).

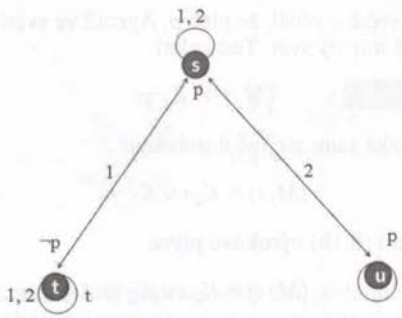
Příklad 8.2. Uvažujme Kripkeho strukturu $\boldsymbol{M} = (S, \pi, K_1, K_2)$ pro jazyk se dvěma agenty A1 a A2 a s jednoprvkovou množinou výrokových konstant $\Phi = \{p\}$. Nechť Kripkeho struktura $\boldsymbol{M}$ obsahuje pouze tři světy $S = \{s, t, u\}$, a to takové, že prvotní formule p platí ve světech s a u, avšak neplatí ve světě t. Tyto požadavky jsou realizovány prostřednictvím vhodné interpretace světů π, která každému světu dává odpovídající pravdivostní ohodnocení, tedy v našem případě hodnoty $\pi(s)(p) = \pi(u)(p) = \text{true}$ a $\pi(t)(p) = \text{false}$.

Dále nechť relace přípustnosti K_1 a K_2 pro agenty A1 a A2 jsou v $\boldsymbol{M}$ definovány takto:

$$K_1 = \{(s, s), (s, t), (t, s), (t, t), (u, u)\},$$
$$K_2 = \{(s, s), (s, u), (u, s), (t, t), (u, u)\}.$$

Znamená to tedy, že zatímco agent A1 neumí rozlišit světy s a t, agent A2 neumí rozlišit svět s od světa u. Celkovou situaci zachycuje obrázek 8.2 se třemi uzly s, t a u, které odpovídají uvažovaným světům.

Hrana spojující světy s a t označená číslem 1 znamená, že Agent1 tyto dva světy neumí rozlišit. Hrana mezi uzly s a t označená číslem 1 má šipky na obou svých koncích díky tomu, že Agent1 ve světě s považuje za možný svět t a naopak i ve světě t považuje za možný svět s. Podobně hrana označená číslem 2 spojuje dva světy, které Agent2 neumí rozlišit, neboť v každém z nich považuje ten druhý za možný. Malé kroužky u všech tří světů označené čísly 1 a 2 naznačují, že relace K_1 a K_2 jsou reflexivní.



Obr. 8.2

Kdyby výrok p odpovídal tvrzení „*prší*", potom by vztahy v obr. 8.2 vypovídaly o tom, že ve světě s prší, ale Agent1 si touto skutečností nemůže být jist, protože ve světě s považuje za možný také svět t, kde výrok p neplatí. Agent1 tedy ve světě s nemůže tvrdit, že ví p. Naopak Agent2 ve světě s ví, že prší, protože kromě s považuje za možný jen svět u a tam p platí také. Můžeme tedy napsat, že platí formule:

$$(\boldsymbol{M}, s) \models \neg K_1 p, \tag{8.1a}$$
$$(\boldsymbol{M}, s) \models K_2 p, \tag{8.1b}$$

a dále pak dokonce $(\boldsymbol{M}, s) \models \neg K_1 p \,\&\, K_2 p$. Podobně jako jsme zdůvodnili (8.1a), můžeme ověřit i $(\boldsymbol{M}, s) \models \neg K_1 \neg p$, což v kombinaci s (8.1a) potvrzuje, že platí

$$(\boldsymbol{M}, s) \models \neg K_1 p \,\&\, \neg K_1 \neg p, \tag{8.1c}$$

Formuli (8.1c) lze zapsat i v ekvivalentním tvaru $(\boldsymbol{M}, s) \models \neg(K_1 p \vee K_1 \neg p)$ či vyjádřit slovy: Agent1 ve světě s neví, jaké je počasí.

Provedeme-li obdobu úvah o světě s a subjektu Agent1 i pro svět t, ověříme, že musí platit $(\boldsymbol{M}, t) \models \neg(K_1 p \vee K_1 \neg p)$. Zkombinujeme-li toto tvrzení s platností vztahu (8.1c), dostáváme

$$(\boldsymbol{M}, s) \models K_1 (\neg(K_1 p \vee K_1 \neg p)), \tag{8.1d}$$

což vyjadřuje fakt, že ve stavu s Agent1 sám o sobě ví, že neví, jaké je počasí.

Ve světě u platí p a Agent1 o tom ví, protože pro něj je u jediný možný svět. Agent2 ve světě u také ví, že platí p, protože ve světě u považuje za další možný svět jen svět s, ve kterém p také platí. Zapíšeme to opět platnými formulemi

$$(\boldsymbol{M}, u) \models K_1 p, \tag{8.2a}$$

$$(\boldsymbol{M}, u) \models K_2 p \text{ a také } (\boldsymbol{M}, u) \models K_1 p \,\&\, K_2 p. \tag{8.2b}$$

Ze vztahů (8.1a) a (8.2a) pak plyne, že ve světě s Agent2 neví, zda Agent1 ví, že platí p, tedy $(\boldsymbol{M}, s) \models \neg K_2 K_1 p$. Podobně ovšem ani neví, že Agent1 neví, zda platí p:

$$(\boldsymbol{M}, \mathrm{s}) \models \neg K_2 \neg K_1 p.$$

Podobně jako Agent1 ve světě u věděl, že platí p, Agent2 ve světě t, kde platí $\neg p$, o tom ví, protože t je pro něj jediný možný svět. Tedy platí

$$(\boldsymbol{M}, t) \models K_2 \neg p.$$

Ze sémantiky disjunkce také samozřejmě dostáváme

$$(\boldsymbol{M}, t) \models K_2 p \vee K_2 \neg p. \tag{8.3a}$$

Podobně z platnosti tvrzení (8.1b) výrokově plyne

$$(\boldsymbol{M}, s) \models K_2 p \vee K_2 \neg p. \tag{8.3b}$$

Ovšem pro Agenta1 je ve světě s svět t jediný další možný, a proto dostáváme

$$(\boldsymbol{M}, s) \models K_1(K_2 p \vee K_2 \neg p). \tag{8.3c}$$

Tvrzení (8.3c) znamená, že ve světě s Agent1 ví, že Agent2 ví, jaké je počasí. Připomeňme si, že naopak sám o sobě Agent1 ví, že neví, jaké je počasí, viz vztah (8.1d).

V kontrastu s tím, i když Agent2 ve světě s ví, že prší, Agent2 v témže světě neví, že Agent1 tento fakt nezná, tj. platí $(\boldsymbol{M}, s) \models \neg K_2(\neg K_1 p)$. Toto tvrzení platí ve světě s díky tomu, že v jednom světě, který Agent2 považuje za možný, jmenovitě ve světě u, Agent1 ví, že prší, ale ve druhém možném světě Agenta2, jmenovitě ve světě s, Agent1 tento fakt neví.

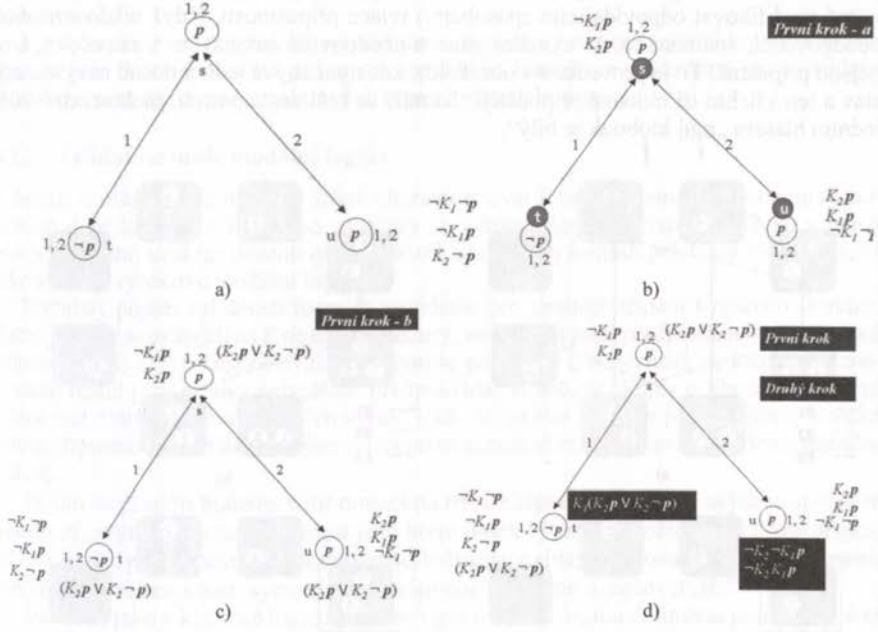
Celou analýzu může shrnout do jediného tvrzení

$$(\boldsymbol{M}, s) \models p \,\&\, \neg K_1 p \,\&\, K_2 p \,\&\, K_1 (K_2 p \vee K_2 \neg p) \,\&\, \neg K_2 \neg K_1 p.$$

Při ověřování podobných formulí je užitečné postupně anotovat vrcholy grafu Kripkeho struktury tak, jak je to naznačeno na obr. 8.3. Začínáme tím, že graficky znázorníme studovanou Kripkeho strukturu, obr. 8.3a). V dalším kroku jsou k jednotlivým vrcholům grafu

připsány formule obsahující zřetězení nanejvýš jednoho modálního operátoru, obr. 8.3b) – krok 1, v kroku 2 formule obsahující zřetězení dvou modálních operátorů, obr 8.3c) atd.

Zkusme využít Kripkeho strukturu pro řešení mírné modifikace hádanky o králi, jeho rádcích a klobouc ích.



Obr. 8.3

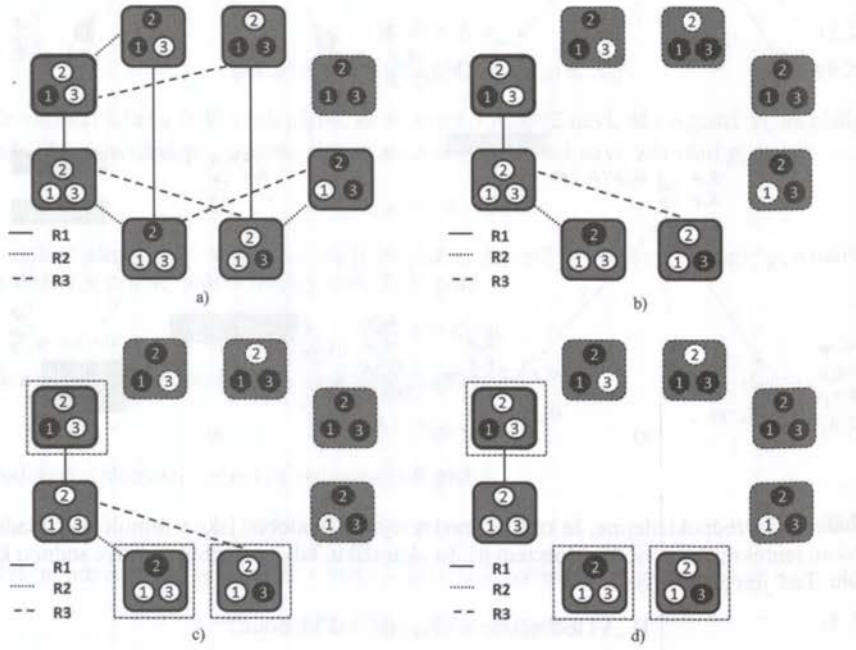
Příklad 8.3: Předpokládejme, že král připraví nový test, podobně jako v minulém příkladu. Ovšem tentokrát král počká s dotazem až do okamžiku, kdy si všichni rádcové sednou ke stolu. Teď jim položí dotaz

D „Ví teď někdo z Vás, jaký má klobouk?"

Žádný z rádců neodpoví – tedy žádný z nich zatím neví, jaký má klobouk. Pak král položí úplně stejnou otázku **D** a reakce rádců je opět stejná. Jak odpovědí jednotliví rádcové, až se král zeptá potřetí?

Obrázek 8.4a) znázorňuje Kripkeho strukturu pro výchozí situaci, v níž jsou rádcům rozdány klobouky, a jediné, čím si mohou být všichni rádcové jisti, je, že určitě žádný z nich nemůže pokládat za přípustnou situaci, kdy všichni tři mají černý klobouk (tolik černých klobouků král neměl). Jak se změní Kripkeho struktura, když na první královu výzvu **D** nikdo z nich neodpoví? Pokud by některý z nich viděl 2 černé klobouky, bylo by mu jasné, že on sám musí mít klobouk bílý. Proto v tomto okamžiku všichni přestanou považovat za možné situace se dvěma černými klobouky. Aktuální graf výsledné Kripkeho struktury vidíme na obr. 8.4b), kde zůstávají pro názornost silně orámované jen ty situace, které alespoň jeden z rádců považuje za možné. Nyní král položí svůj dotaz podruhé a opět nikdo nic

neví. Kdyby například měl první rádce černý klobouk a druzí dva bílý, musel by tento rádce odpovědět „nevím". Ovšem druhý a třetí rádce v tomto stavu nepřipouštějí žádnou alternativu, neboť z tohoto stavu v grafu z obr. 8.4b) nevede žádná hrana odpovídající jejich relacím přípustnosti. Tedy druhý i řetí rádce by mohli oznámit, že znají barvy svých kloboučků. Protože se tak nestalo, mohou tento stav v tomto okamžiku všichni vyloučit. Podobné stavy jsou v této struktuře ještě tři a jsou na obr. 8.4c) zvýrazněny rámečkem. Samozřejmě, že je nutné modifikovat odpovídajícím způsobem i relace přípustnosti. Když nikdo z mudrců neodpověděl, znamená to, že aktuální situace neodpovídá ani stavům v rámečcích, a ty nejsou přípustné. To je provedeno v obr. 8.4d), kde nyní zbývá jediný dosud nevyloučený stav a ten všichni tři rádcové připouštějí. Jakmile se král zeptá potřetí, mohou odpovědět jedním hlasem „můj klobouk je bílý".



Obr. 8.4

Předchozí příklady ukazují, že jazyk výrokové modální logiky s modálními operátory nabízí zajímavé výrazové prostředky a že Kripkeho struktury představují užitečný nástroj pro odvozování tvrzení, která platí v možných světech.

Samozřejmě, že není těžké uvést příklady tvrzení, která naopak ve výrokové modální logice vyjádřit nemůžeme. Stačí si vzpomenout třeba na vztahy, při jejichž popisu se opíráme o kvantifikátory, jako když chceme říci, že: „*Mařenka ví, že každá houba v lese má své jméno*," nebo že: „*Mařenka umí pojmenovat (zná jména pro) všechny houby v lese.*" Oběma těmto tvrzením odpovídají různé formule predikátové modální logiky:

$$(\forall x)[\text{houba}(x) \rightarrow K_{\text{Mařenka}}((\exists y)\text{jméno}(x, y))],$$
$$(\forall x)[\text{houba}(x) \rightarrow (\exists y)(K_{\text{Mařenka}}\,\text{jméno}(x, y))].$$

Jazyk by v tomto případě podobně jako v klasické predikátové logice musel obsahovat vedle modalit i proměnné pro individua, unární predikát houba/1 či binární predikát jméno/2, a nabízet možnost kvantifikovat proměnné pro individua. Predikátová modální logika je nepochybně silnější než výroková modální logika, ale bez jejích subtilních kouzel je možné se často obejít. Jistě není bez zajímavosti, že v samotné definici sémantiky operátoru znalosti ve výrokové modální logice je patrný náznak univerzální kvantifikace na množině možných světů.

V této kapitole se soustředíme na studium vlastností výrokové modální logiky, která se ukazuje jako dostatečně silná pro mnoho aplikací. Naznačíme především, jak se buduje formální aparát pro dokazování nových tvrzení v modální logice.

8.1.2 Validní formule modální logiky

Položme si otázku, zda můžeme nějak charakterizovat formule platné ve všech možných světech dané konkrétní Kripkeho struktury či dokonce formule platné ve všech světech všech Kripkeho struktur daného typu. Některé z takových formulí pak bude možné použít jako axiomy výrokové modální logiky.

Potřebný pojem validnosti formule zavedeme pro jemnější třídění Kripkeho struktur. Zatím jsme se soustředili na Kripkeho struktury, ve kterých jsme předpokládali, že relacemi přípustnosti K_i jsou relace ekvivalence. Nyní se pokusíme o zobecnění, ve kterém o vlastnostech relací přípustnosti nebudeme předpokládat vůbec nic. Půjde o libovolné binární relace nad množinou uvažovaných světů. Třídu Kripkeho struktur pro n agentů, v nichž relace přípustnosti jsou libovolnými relacemi na množině možných světů, budeme označovat $\mathcal{M}_n$.

Pokud se později budeme chtít omezit na třídu Kripkeho struktur s nějakou speciální vlastností, zvýrazníme tuto vlastnost jako horní index výchozí množiny $\mathcal{M}_n$. Proto případ, když relacemi přípustnosti jsou ekvivalence, tedy relace charakterizované třemi vlastnostmi, kterými jsou reflexivnost, symetrie a tranzitivnost, budeme označovat $\mathcal{M}_n^{\mathrm{rst}}$.

Podobně jako v klasické logice budeme i pro modální logiku definovat pojmy, které se vztahují k platnosti formule ve všech světech dané struktury nebo dané množiny struktur:

- „Říkáme, že formule A je *validní ve struktuře* $\boldsymbol{M}$ ze třídy $\mathcal{M}_n$, jestliže A platí ve všech světech struktury $\boldsymbol{M}$" a píšeme $\boldsymbol{M} \models A$.
- „Říkáme, že formule A *je validní ve třídě* $\mathcal{M}_n$, je-li validní ve všech strukturách třídy $\mathcal{M}_n$", a píšeme $\models A$.
- „Říkáme, že formule A je *splnitelná ve struktuře* $\boldsymbol{M}$ ze třídy $\mathcal{M}_n$, jestliže A platí alespoň v jednom světě struktury $\boldsymbol{M}$".
- „Říkáme, že formule A je *splnitelná ve třídě* $\mathcal{M}_n$, jestliže A je splnitelná alespoň v jedné struktuře třídy $\mathcal{M}_n$".

Podobně můžeme definovat validnost a splnitelnost i pro třídu $\mathcal{M}_n^{\mathrm{rst}}$. Pokud bude jasné, o jakou třídu jde, můžeme používat stejný symbol $\models$ bez zvláštního označení.

Z uvedených definic je zřejmé, že pro libovolnou strukturu $\boldsymbol{M}$ platí, že formule A je v $\boldsymbol{M}$ validní, právě když formule $\neg A$ není v $\boldsymbol{M}$ splnitelná. Analogicky, formule A není validní v nějaké třídě struktur, jestliže není validní v nějaké struktuře dané třídy. Uveďme několik vlastností právě zavedených pojmů.

VĚTA 8.1. Pro každou strukturu $\boldsymbol{M}$ třídy $\mathcal{M}_n$ platí následující tvrzení:

a) Všechny instance axiomů výrokové logiky jsou v ní validní.

b) Pokud formule A i $A \rightarrow B$ jsou v $\boldsymbol{M}$ validní, pak musí být v $\boldsymbol{M}$ validní také formule B.
c) Nechť A je validní formule $\boldsymbol{M}$, pak také formule $K_i A$ je v $\boldsymbol{M}$ validní.
d) Nechť A a B jsou libovolné formule. Pak je v $\boldsymbol{M}$ validní formulí, někdy nazývaná *Kripkeho formule*, $(\mathrm{K}_i A \,\&\, \mathrm{K}_i(A \rightarrow B)) \rightarrow K_i B$.

Důkaz. Tvrzení (a) i (b) jsou přirozeným důsledkem toho, že logické spojky jsou v definici $\models$ interpretovány stejně jako ve výrokové logice.

Abychom se přesvědčili o platnosti (c), uvažujme libovolnou strukturu $\boldsymbol{M}$ ze třídy $\mathcal{M}_n$ a libovolný svět s této struktury. Protože předpokládáme, že A je validní formule struktury $\boldsymbol{M}$, musí pro každý svět t této struktury být A pravdivá, tj. platit $(\boldsymbol{M}, t) \models A$. Protože je A pravdivá v každém světě t struktury $\boldsymbol{M}$, musí být pravdivá i ve všech světech, které agent i považuje v s za možné (světy, které agent i považuje v s za možné, jsou podmnožinou všech světů struktury $\boldsymbol{M}$). Proto tedy $(\boldsymbol{M}, s) \models K_i A$.

Při důkazu tvrzení (d) budeme postupovat podobně jako v předchozím případě. Uvažujme libovolnou strukturu $\boldsymbol{M}$ ze třídy $\mathcal{M}_n$ a libovolný svět s této struktury. Je třeba rozlišit dva případy. Nejprve se soustředíme na ty světy s, ve kterých je předpoklad implikace tvořící Kripkeho formuli splněn, tj. $(\boldsymbol{M}, s) \models K_i A \,\&\, K_i(A \rightarrow B)$. To znamená, že v každém světě t takovém, že $(s, t) \in K_i$, platí jak $(\boldsymbol{M}, t) \models A$, tak i $(\boldsymbol{M}, t) \models A \rightarrow B$. Z definic vztahů $\models$ a $\rightarrow$ ovšem vyplývá, že v uvažovaném světě t musí platit i formule B, tj. $(\boldsymbol{M}, t) \models B$, a tedy i $(\boldsymbol{M}, s) \models K_i B$. Kripkeho formule je v takovém světě s ověřena. V případě světa s, v němž není splněn předpoklad implikace v Kripkeho formuli, je z definice implikace zřejmé, že ve světě s Kripkeho formule musí platit, neboť z nepravdy plyne cokoliv.

8.1.3 Axiomatický systém a jeho vlastnosti

Právě vlastnosti, které jsme ověřili v předchozím odstavci, hrají významnou roli při zavedení nejjednoduššího axiomatického systému $\mathcal{K}_n$, který nabízí nástroje pro práci s formulemi jazyka modální logiky, v němž je standardní jazyk výrokové logiky L_Φ obohacen o modální operátory K_i pro $1 \leq i \leq n$. Axiomatický systém $\mathcal{K}_n$ má dvě třídy axiomů definované pomocí schémat formulí:

– třída $\mathcal{A}$ obsahuje všechny *instance axiomů výrokové logiky*,
– třída $\mathcal{K}$ obsahuje pro libovolné formule A, B uvažovaného jazyka odpovídající Kripkeho formuli, tj. obsahuje všechny formule tvaru $(K_i A \,\&\, \mathrm{K}_i(A \rightarrow B)) \rightarrow K_i B$;

a dvě odvozovací pravidla:

– *Pravidlo modus ponens* (zkráceně MP), které dovoluje ze dvou formulí tvaru A a $A \rightarrow B$ odvodit novou formuli B.
– *Pravidlo generalizace* umožňující z formule A odvodit formuli $K_i A$.

Pojem důkazu v daném axiomatickém systému je obdobou definice používané v klasické logice, viz (Štěpánková a Štěpánek, 1995). Tedy *důkaz formule* α v axiomatickém systému $\mathcal{K}_n$ tvoří konečná posloupnost formulí $\varphi_1, \ldots, \varphi_n$ taková, že φ_n je právě formule α a pro každý člen φ_i této posloupnosti platí:

– buď jde o prvek jedné ze tříd $\mathcal{A}$ nebo $\mathcal{K}$,
– nebo členu φ_i předcházejí v posloupnosti dva jiné členy φ_k a φ_h (tedy $k < i$ a také $h < i$) takové, že z φ_k a φ_h vznikne φ_i aplikací pravidla modus ponens,
– nebo prvek φ_i předchází v posloupnosti jiný prvek φ_k (tedy $k < i$) takový, že formule φ_i je výsledkem aplikace pravidla generalizace na prvek φ_k.

Při hodnocení vlastností odvozovacích pravidel nás jistě zajímá, zda aplikací odvozovacího pravidla na validní formule vzniknou opět validní formule. Má-li odvozovací pravidlo tuto vlastnost, říkáme, že jde o *korektní odvozovací pravidlo*.

Bod (b) z věty 8.1 říká tedy, že odvozovací pravidlo *modus ponens* je korektní a bod (c) tvrdí totéž o pravidlu generalizace. Body (a) a (d) pak ověřují, že všechny formule ve třídách před $\mathcal{A}$ a $\mathcal{K}$ jsou validní. Díky těmto vlastnostem se tedy můžeme lehce přesvědčit indukcí, že každá formule dokazatelná v axiomatickém systému $\mathcal{K}_n$ je validní vzhledem k $\mathcal{M}_n$. Tato vlastnost se označuje jako *korektnost axiomatického systému*.

Dá se dokonce ukázat, že $\mathcal{K}_n$ je nejen korektní, ale i *úplný axiomatický systém* pro třídu $\mathcal{M}_n$. To znamená, že ke každé formuli φ validní ve třídě $\mathcal{M}_n$ je možné v axiomatickém systému $\mathcal{K}_n$ dokonce sestrojit její důkaz. Právě tyto dvě vlastnosti hrají klíčovou roli pro automatizaci dokazování v axiomatických systémech. Díky tomu, že existuje korektní úplný axiomatický systém pro modální logiku, je možné používat pro dokazování v uvedených axiomatických systémech podobné techniky jako v případě klasické logiky, viz (Urban a kol., 2013). Při ověřování úplnosti uvažovaného axiomatického systému se postupuje obdobně jako ve výrokové logice – stačí ukázat, že není-li φ dokazatelná formule, pak lze sestrojit strukturu $\boldsymbol{M}$ ze třídy $\mathcal{M}_n$, ve které je splnitelná formule $\neg\varphi$ (a tedy formule φ není validní ve třídě $\mathcal{M}_n$). Tvrzení o úplnosti uvažovaného axiomatického systému má zásadní význam, neboť ukazuje, že tento systém plně charakterizuje právě všechny validní formule. Přesto toto tvrzení předkládáme čtenáři bez důkazu, neboť ten má poněkud technický charakter. Čtenáři, kterého zajímá důkaz tohoto tvrzení, doporučujeme knihu (Fagin, Halpern, Moses, Vardi, 2003), která vedle velmi důkladného úvodu do teoretických základů modální logiky pro více agentů nabízí i řadu dobře zvolených příkladů.

Pokud budeme předpokládat, že relace přípustnosti vedle již uvedených vlastností splňují i další předpoklady, omezí se tím třída uvažovaných Kripkeho struktur, a to může vést k tomu, že se validními stanou ještě další formule. Následující věta upozorňuje na formule a odvozovací pravidla, která jsou korektní ve strukturách s relacemi přípustnosti, jež mají všechny vlastnosti ekvivalence.

VĚTA 8.2. Pro všechny formule jazyka L_n a všechny struktury $\boldsymbol{M}$ ze třídy $\mathcal{M}^{\mathrm{rst}}{}_n$ platí:

a) $\boldsymbol{M} \models (K_i A \,\&\, K_i (A \to B)) \to K_i B$,

b) Je-li $\boldsymbol{M} \models A$, potom $\boldsymbol{M} \models K_i A$,

c) Pokud formule A i $A \to B$ jsou v $\boldsymbol{M}$ validní, pak musí být v $\boldsymbol{M}$ validní také formule B,

d) $\boldsymbol{M} \models K_i A \to A$,

e) $\boldsymbol{M} \models K_i A \to K_i K_i A$,

f) $\boldsymbol{M} \models \neg K_i A \to K_i \neg K_i A$.

Tato věta upozorňuje na hlubší souvislosti vztahu mezi vlastnostmi relací přípustnosti a vhodnou volbou axiomatického systému. Právě proto doporučujeme čtenáři, aby si tento důkaz řádně prostudoval.

Důkaz. Tvrzení (a), (b) a (c) jsou zde uvedena spíš jen pro pořádek. Dokázali jsme je již ve větě 1 pro libovolnou strukturu $\boldsymbol{M}$ ze třídy $\mathcal{M}_n$. Musejí tedy platit i pro struktury ze třídy $\mathcal{M}_n{}^{\mathrm{rst}}$. Dokážeme postupně zbývající tři tvrzení, která obecně pro všechny struktury ze třídy $\mathcal{M}_n$ neplatí.

Začněme tvrzením (d) a uvažujme libovolný svět s v $\boldsymbol{M}$ ze třídy $\mathcal{M}_n{}^{\mathrm{rst}}$. Předpokládejme, že $(\boldsymbol{M}, s) \models K_i A$. To znamená, že pro každý svět t takový, že $(s, t) \in K_i$, musí platit $(\boldsymbol{M}, t) \models A$.

Ovšem pak to musí platit i pro stav s, neboť K_i je také reflexivní relace, a tedy nepochybně $(s, s) \in K_i$. Ověřili jsme, že pokud $(\boldsymbol{M}, s) \models K_i A$, pak platí i $(\boldsymbol{M}, s) \models A$, a tedy i $(\boldsymbol{M}, s) \models K_i A \rightarrow A$. V případě, že neplatí $(\boldsymbol{M}, s) \models K_i A$, je tvrzení $(\boldsymbol{M}, s) \models K A \rightarrow A$ triviálním důsledkem definic $\models$ a $\rightarrow$.

Chceme-li dokázat tvrzení (e), soustředíme se, stejně jako v předchozím případě, na netriviální část tohoto důkazu a budeme uvažovat libovolný svět s v $\boldsymbol{M}$ takový, že $(\boldsymbol{M}, s) \models K_i A$. Potřebujeme tedy ověřit, že platí $(\boldsymbol{M}, s) \models K_i K_i A$. Zajímá nás tedy, zda pro libovolné t takové, že $(s, t) \in K_i$, platí $(\boldsymbol{M}, t) \models K_i A$. Kdyby tomu tak nebylo, musel by v $\boldsymbol{M}$ existovat svět u takový, že $(t, u) \in K$, ve kterém neplatí $(\boldsymbol{M}, u) \models A$. Ovšem vzhledem k tomu, že relace K_i je tranzitivní, musí platit nejen $(t, u) \in K_i$, ale také $(s, u) \in K_i$. Pokud neplatí $(\boldsymbol{M}, u) \models A$, neplatí ani výchozí předpoklad $(\boldsymbol{M}, s) \models \mathrm{K}_i A$. Tedy svět u s uvedenými předpoklady nemůže existovat a pro libovolné t takové, že $(s, t) \in K_i$, platí $(\boldsymbol{M}, t) \models K_i A$. Tak jsme ověřili, že platí-li $(\boldsymbol{M}, s) \models K_i A$, pak platí také $(\boldsymbol{M}, s) \models K_i K_i A$.

Dokažme tvrzení (f) – mějme libovolný svět s takový, že $(\boldsymbol{M}, s) \models \neg \mathrm{K}_i A$. Potom pro nějaké u takové, že $(s, u) \in K_i$, platí $(\boldsymbol{M}, u) \models \neg A$. Naším úkolem je ověřit, že pro libovolný svět t takový, že $(s, t) \in K_i$, platí $(\boldsymbol{M}, t) \models \neg \mathrm{K}_i A$. Pokud se nám podaří ověřit, že $(t, u) \in K_i$, je náš úkol splněn. Nejprve využijeme symetrie relace K_i, díky níž zjistíme, že platí také $(t, s) \in K_i$. Dál nám pomůže tranzitivnost K_i a předpoklad $(s, u) \in K_i$ spolu s právě ověřeným vztahem $(t, s) \in K_i$, abychom se přesvědčili, že $(t, u) \in K_i$. Tedy skutečně pro všechna uvažovaná t platí $(\boldsymbol{M}, t) \models \neg K_i A$, což má za důsledek $(\boldsymbol{M}, s) \models K_i \neg K_i A$.

Povšimněme si, že k důkazu (d) postačil předpoklad reflexivnosti relace K_i, k důkazu (e) předpoklad tranzitivnosti a pro důkaz (f) bylo třeba předpokládat kombinaci toho, že K_i je současně symetrická i tranzitivní. Věta 8.2 upozorňuje na další formule validní vzhledem ke třídě $\boldsymbol{\mathcal{M}}_n^{\mathrm{rst}}$. Tyto formule se nabízejí jako vhodní kandidáti na další axiomy (přesněji axiomová schémata, neboť za A lze dosadit libovolnou dobře utvořenou formuli uvažovaného jazyka):

$$\begin{aligned} \mathbf{4}_n \quad & K_i A \rightarrow K_i K_i A, \\ \mathbf{5}_n \quad & \neg K_i A \rightarrow K_i \neg K_i A, \\ \mathbf{T}_n \quad & K_i A \rightarrow A. \end{aligned}$$

Pokud zavedeme třídu $\boldsymbol{\mathcal{M}}_n^{\mathrm{r}}$ všech struktur, v níž relace K_i jsou reflexivní, a třídu $\boldsymbol{\mathcal{M}}_n^{\mathrm{rt}}$, kde všechny relace K_i jsou reflexivní a tranzitivní, pak věta 2 ukazuje, že:
– axiomy $\mathbf{T}_n$ jsou validní vzhledem ke třídě $\boldsymbol{\mathcal{M}}_n^{\mathrm{r}}$,
– axiomy $\mathbf{4}_n$ jsou validní vzhledem ke třídě $\boldsymbol{\mathcal{M}}_n^{\mathrm{rt}}$ a
– axiomy $\mathbf{5}_n$ jsou validní vzhledem ke třídě $\boldsymbol{\mathcal{M}}_n^{\mathrm{rst}}$.
Axiomy $\mathbf{4}_n$ se někdy označují jako axiomy *pozitivní introspekce*: „Když Agent něco ví, pak ví, že o tom ví," a naopak o axiomech $\mathbf{5}_n$ se mluví jako o axiomech *negativní introspekce*: „Když Agent něco neví, pak ví, že o tom neví."

Axiomy $\mathbf{T}_n$ se nazývají *axiomy pravdy* nebo *axiomy znalostí*. Jde o tvrzení tvaru „co Agent ví, je pravda", odtud také písmeno **T**. Axiom $\mathbf{T}_n$ je velmi silný, neboť tvrdí, že vše, co Agent ví, musí být validní formulí. Proto se pro některé aplikace nahrazují slabším axiomem

$$\mathbf{D}_n \neg K_i \mathit{false}.$$

Axiom $\mathbf{D}_n$ se omezuje na to, že požaduje, aby „to, co Agent ví, nemohlo poškodit bezespornost". Proto se $\mathbf{D}_n$ označuje jako *axiom bezespornosti*.

Uvedené axiomy nabízejí různá rozšíření výchozího systému modální logiky $\mathcal{K}_n$, mohou se vzájemně kombinovat, a tak získáváme další systémy modální logiky. Například teorie $\mathcal{K}\mathbf{T4}_n$ vznikne přidáním axiomů $\mathbf{T}_n$ a $\mathbf{4}_n$ k modálnímu systému $\mathcal{K}_n$. Podobně vznikají teorie $\mathcal{K}\mathbf{D}_n$, $\mathcal{K}\mathbf{D45}_n$, $\mathcal{K}\mathbf{T}_n$ atd. Vedle právě uvedeného označení se pro tyto systémy někdy používají i alternativní jména, která jsou výsledkem jednoduché modifikace vžitých tradičních názvů systémů modální logiky pro případ, kdy pracujeme s více agenty. Například jméno

$\mathbf{S4}_n$ označuje systém $\mathcal{K}\mathbf{T4}_n$,
$\mathbf{S5}_n$ označuje systém $\mathcal{K}\mathbf{T45}_n$.

Index n se obvykle vynechává, pokud pracujeme s pevným počtem agentů. Logiky $\mathbf{S4}_n$ a $\mathbf{S5}_n$ byly původně navrženy pro $n = 1$. V tomto případě mají navíc některé další vlastnosti oproti logikám pro $n > 1$, viz (Hintikka, 1962). Není těžké se přesvědčit, že Kripkeho struktury pro uvedené teorie lze velmi dobře charakterizovat pomocí vlastností relací přípustnosti – přesný vztah popisuje následující věta, kterou uvádíme bez důkazu.

VĚTA 8.3. Pro formule jazyka L_n platí:
$\mathcal{K}\mathbf{T}_n$ je korektní a úplná axiomatizace vzhledem ke třídě $\mathcal{M}_n^{\mathrm{r}}$,
$\mathbf{S4}_n$ je korektní a úplná axiomatizace vzhledem ke třídě $\mathcal{M}_n^{\mathrm{rt}}$,
$\mathbf{S5}_n$ je korektní a úplná axiomatizace vzhledem ke třídě $\mathcal{M}_n^{\mathrm{rst}}$,

Pokud chceme nahradit axiom pravdy $\mathbf{T}_n$ slabším axiomem $\mathbf{D}_n$, je třeba při budování modelů zmírnit požadavky na vlastnosti relace přípustnosti. Ukáže se, že tato relace už nemusí být reflexivní, naopak významnou roli budou hrát relace *eukleidovské*, tj. takové, pro které platí

$$((s, t) \in K_i \,\&\, (s, u) \in K_i) \rightarrow (t, u) \in K_i,$$

a relace *seriální*, tedy takové, že pro každý svět s existuje nějaký svět t, pro něhož platí $(s, t) \in K_i$.

Označíme-li $\mathcal{M}_n^{\mathrm{elt}}$ třídu všech Kripkeho struktur, ve kterých jsou relace přípustnosti eukleidovské, seriální a tranzitivní, dá se dokázat, že

$\mathcal{K}\mathbf{D45}_n$ je korektní a úplná axiomatizace vzhledem ke třídě $\mathcal{M}_n^{\mathrm{elt}}$.

Tato logika je slabší verzí $\mathbf{S5}_n$, v níž je axiom pravdy nahrazen axiomem bezespornosti $\mathbf{D}_n$. Tento systém odpovídá situaci, kdy operátor $\Box_i$ není interpretován jako znalost agenta i, ale jako jeho přesvědčení (belief). V takovém případě označujeme použitý modální operátor *přesvědčení* jako B_i. Je zřejmé, že nemůžeme přijmout axiom pravdy, protože $B_i A \rightarrow A$ již nemusí být validní formule: to, o čem je agent přesvědčen, nemusí být v daném světě pravda. Tomu odpovídá i formulace axiomu bezespornosti. Lze tedy říci, že axiom $\mathbf{T}_n$ charakterizuje rozdíl mezi skutečnou znalostí a přesvědčením, které popisuje axiom $\mathbf{D}_n$.

V dosavadním výkladu jsme nejprve interpretovali operátor nutnosti $\Box_n$ jako operátor znalosti K_n. Teď jsme si povšimli, že pomocí modální logiky lze formalizovat i úvahy zabývající se tím, o čem jsou agenti přesvědčeni, ale neví to jistě. Jsou studovány i logiky dokazatelnosti, které interpretují $\Box_n$ pro $n = 1$ jako dokazatelnost v nějakém formálním systému. V takovém případě výrokové konstanty p, q, r zastupují libovolné formule nějaké matematické teorie a modální formule $\Box p$ znamená, že formule p je dokazatelná (obvykle v Peanově aritmetice).

Jsou možné i modální logiky se zcela jiným pojetím nutnosti. Důležitými příklady jsou temporální logiky, které vztahují nutnost k okamžikům probíhajícího času, a proto mají velký význam při studiu vlastností dynamických systémů. Temporálním logikám bude věnován oddíl 8.2 této kapitoly.

8.1.4 Společná znalost (Common Knowledge)

Až dosud jsme se zabývali tím, co vědí jednotliví agenti. Ovšem v běžném životě se často musíme spolehnout na to, že některou informaci má každý agent z dané skupiny agentů G nebo dokonce, že některá tvrzení představují společnou znalost pro všechny subjekty z G. Pro tyto účely se jazyk modální logiky dále doplňuje o další modality E_G („*ve skupině G každý ví*") a C_G („*společnou znalostí všech agentů ve skupině G je*"), které popíšeme pomocí už známých modalit K_i.

Nechť φ je nějaká formule zvoleného jazyka modální logiky, pak pro libovolnou Kripkeho strukturu S a její svět s platí

$$(S, s) \models E_G\varphi \quad \text{právě tehdy, když} \quad (\forall g \in G)\,((S, s) \models K_g\varphi).$$

Samozřejmě, že má význam hovořit i o vícenásobné aplikaci modality E_G tak, jak tomu je třeba ve větě „*každý ve skupině G ví, že každý v této skupině ví* φ", kterou bychom formálně zapsali jako $E_G(E_G\varphi)$. Protože modalita „*každý ví*" se může aplikovat na tutéž formuli mnohokrát a vznikají tak formule nové, jejichž pravdivostní hodnoty se od sebe mohou vzájemně lišit, zavádí se přehlednější notace, ze které je zřejmé kolikrát ke zřetězení došlo: Formule $E_G^0\varphi$ se chápe jako ekvivalent formule φ a pro libovolné přirozené číslo k je formule $E_G^{k+1}\varphi$ zkratkou za $E_G(E_G^k\varphi)$.

Pomocí této nové notace jsme připraveni definovat $C_G\varphi$, tedy společnou znalost tvrzení φ pro skupinu agentů G, takto: v libovolné Kripkeho struktuře S a jejím světě s platí $C_G\varphi$ právě tehdy, když ve světě s struktury S platí $E_G^k\varphi$ pro všechna $k = 1, 2, \ldots$

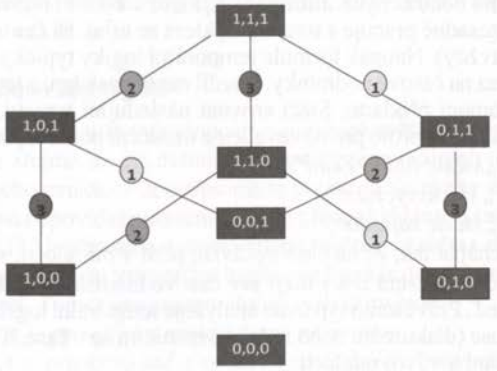
Rozdíl mezi společnou znalostí a tím, že každý v nějaké skupině ví, budeme ilustrovat v následujícím příkladu.

Příklad 8.4: Nechť tři malí sourozenci sedí kolem stolu, kde je krabice s čokoládou, ze které si mohou brát. Nikdo z nich nemá ubrousek ani zrcátko, takže žádný nemůže zkontrolovat, zda se při mlsání neušpinil. Všichni tři jsou zvyklí, že si ze sebe často vzájemně dělají legraci, takže nemohou zjistit, zda jsou špinaví dotazem u některého ze sourozenců. Najednou přijde jejich tatínek a řekne nahlas, aby to všechny děti slyšely „*Tady je někdo špinavý. Za chvíli se zeptám každého z vás, zda je ušmudlaný či nikoliv. Odměním toho z vás, který je špinavý a přitom na to přijde sám čistě silou své racionální úvahy. Teď přemýšlejte.*"

Jsou-li například právě dvě děti u stolu špinavé, určitě platí, že „*každé dítě ví, že je mezi nimi někdo špinavý*", ale neplatí že „každé dítě ví, že každé dítě ví, že je mezi nimi někdo špinavý". Stačí si uvědomit, že první špinavé dítě neví, zda druhé špinavé dítě ví, že je mezi nimi někdo špinavý (zda vidí někoho špinavého). Teprve díky tatínkovu hlasitému oznámení všichni vědí, že všichni vědí, ... a získávají tak společnou znalost, kterou před tím neměli.

Mohou děti tuto společnou znalost nějak využít? Obrázek 8.5 znázorňuje Kripkeho strukturu pro naši úlohu, která obsahuje 2^3 různých světů spojených podle toho, jak je jednotlivé děti připouštějí po tatínkově prohlášení „*Tady je někdo špinavý.*" Povšimněme si, že svět (0, 0, 0), ve kterém jsou všechny tři děti čisté, už nikdo nepřipouští. Předpokládejme, že

je špinavé jen jedno z dětí, třeba dítě 1 – v tom případě jde o svět (1, 0, 0) a dítě 1 v něm nemůže připouštět žádnou alternativu. Dítě 1 tedy může radostně odpovědět, až bude tázáno: „*Jsem to já!*". Podobná úvaha platí i v případě, že jediným šmudlou je dítě 2 nebo dítě 3. Pokud všechny děti odpověděly, že neví, musely také dojít k závěru, že mezi nimi není jen jeden šmudla, ale že je jich víc. Teď otec řekne „*Dobře. Zkusíme to ještě jednou. Zamyslete se a já se za chvíli zeptám znovu.*" A děti už vědí, že nejméně dvě z nich jsou špinavé (je to jejich společná znalost). Je-li mezi nimi někdo, kdo vidí jen jednoho špinavého sourozence, musí mu být jasné, že on sám je také špinavý. Když se otec zeptá, buď dvě z dětí oznámí, že jsou špinavé nebo všechny řeknou nevím. V tom případě v Kripkeho struktuře přestanou děti považovat za přípustné všechny světy se dvěma šmudly a zbývá poslední svět, kde není čistý nikdo. Mohou tedy jasně odpovědět. Společná znalost jim pomohla odvodit něco, co předtím nikdo z nich nevěděl.



Obr. 8.5 Kripkeho struktura pro tři ušmudlané děti poté, co otec konstatoval „*Tady je někdo špinavý.*"

V odstavci 8.1.3 jsme viděli, že vlastnosti modalit typu K_i lze charakterizovat pomocí souboru axiomů, které tvoří korektní a úplný formální systém. Podobně lze přistoupit i k modalitám E_G a C_G a popsat jejich vlastnosti pomocí korektního a úplného formálního systému, viz (Fagin a spol., 1995 a 2003).

Společná znalost hraje významnou roli v řadě reálných situací – reprezentuje přijaté normy chování typu „*jezdíme na pravé straně silnice, předjíždění auto míjíme zleva*". Bez společné znalosti se neobejdou například agenti, kteří chtějí koordinovaně provést nějakou akci. Otázka, jak může skupina agentů získat společnou znalost, je tedy nad míru důležitá. A právě nástroje modální logiky na ni mohou pomoci hledat odpověď. Lze ukázat, že pouhým posíláním zpráv od jednoho agenta ke druhému v prostředí, kde není zaručena rychlost doručení zprávy, nebo kde zprávu může příležitostně ovlivnit šum informačního kanálu, nelze společnou znalost ztratit ani získat. Přesně toto tvrzení vyslovuje „věta o zachování společné znalosti pro systémy s asynchronním předáváním zpráv", jejíž rigorózní důkaz nabízí skvělá monografie (Fagin a spol., 1995 a 2003). V téže publikaci je podrobně rozebrán i další zajímavý problém, kterému byla věnována velká pozornost v devadesátých letech minulého století, a to hledání protokolu, který by umožnil získat společnou znalost v systémech, v nichž komunikaci zajišťuje pouhá výměna zpráv pro případ, že předávání zpráv je sice spolehlivé a okamžité, ale kde naopak u některých agentů může dojít k různým

druhům chyb či selhání. Dá se ukázat, že takový systém o n agentech, z nichž nanejvýš u t může dojít k selhání, může za určitých podmínek prostřednictvím výměny zpráv vytvořit společnou znalost. Nutná a postačující podmínka pro získání společné znalosti záleží v tomto případě na velikosti poměru mezi n a t: stačí totiž, aby skupina selhávajících byla ve výrazné menšině (přesněji $n > 3t$).

8.2 Temporální logiky

Podobně jako modální logiky jsou temporální logiky odvětvím symbolické logiky, které se věnuje tvrzením, jejichž pravdivost se mění – v tomto případě závisí na čase. Pohyb v čase je znázorněn průchodem různými světy. To kontrastuje s klasickou logikou, kde libovolněkrát vyslovené tvrzení musí mít vždy stejnou pravdivostní hodnotu a nesmí záležet na způsobu opakování ani na dodatečných informacích, které v tvrzení nejsou obsaženy.

Klasická logika zásadně pracuje s tvrzeními, která se nijak na čas neodkazují (vše platí teď nebo dokonce navždy). Naopak formule temporální logiky typicky obsahují (explicitní nebo implicitní) odkaz na časové podmínky. Rozdíl mezi klasickou a temporální logikou lze nejlépe ilustrovat pomocí příkladu. Stačí srovnat následující tvrzení, ke kterým se ještě v dalším výkladu vrátíme, a proto pro ně zavedeme označení pomocí písmen řecké abecedy:

– α nechť znamená „*Měsíc obíhá Zemi*“,
– β nechť znamená „*Měsíc vychází*“,
– γ nechť znamená „*Měsíc zapadá*“.

Tvrzení α můžeme chápat tak, že na čase nezávisí, platí v minulosti, v současnosti i v budoucnosti. Naproti tomu tvrzení β a γ mají své časové hledisko a vztahují se k implicitní časové podmínce – teď. Právě tento typ úvah analyzuje temporální logika, a proto se vztahuje k univerzům diskuse (diskurzům nebo světům) měnícím se v čase. Temporální logika se soustřeďuje na chování a vývoj událostí.

Zatímco formule klasické logiky mohou popisovat statické stavy a vlastnosti, formule temporální logiky mohou popisovat sekvence změn stavů a změn chování. Temporální logika – stejně jako modální logika – je rozšířením klasické výrokové nebo predikátové logiky o nové operátory, které zavádějí temporální modality. V tomto oddílu se budeme opět zabývat jen výrokovým případem.

Základní temporální operátory jsou nejméně dva:

– $\Box$ vyjadřující vždy,
– $\Diamond$ označující někdy.

U obou se předpokládá, že se vztahují k budoucnosti, tj. ke světům, do nichž by se systém časem mohl dostat. Typické složené formule jsou podobné těmto příkladům:

– $\Diamond\beta$, která vyjadřuje tvrzení „*Měsíc někdy vyjde*“,
– $\Box\Diamond\beta$ s významem „*Měsíc bude vycházet znovu a znovu*“,
– $\Box(\beta \rightarrow \Diamond\gamma)$ s významem „*Vždy po východu měsíce bude někdy následovat jeho západ*“.

Formule δ je pravdivá, jestliže Měsíc vychází teď, nebo vyjde v některém pozdějším časovém bodě. Formule ε ukazuje, že kombinace temporálních operátorů může vyjadřovat složitější časové podmínky, například pro kombinaci modalit $\Box\Diamond$(vždy-někdy) je v přirozeném jazyce asi nejvhodnější použít výraz „znovu a znovu“. Formule ϕ je vyjádřením následnosti dějů: předpoklad (podmínka) β povede někdy v budoucnu k následku γ (někdy označovaném jako následující podmínka).

Uvedené příklady naznačují, že temporální logika může být použita k popisu a specifikaci procesů a systémů, které procházejí různými stavy. K tomu ji stačí doplnit

o některé další operátory. Před tím je nutné říci, jaký čas budeme mít na mysli. Z různých možností vybereme tu nejjednodušší. Budeme předpokládat, že každý proces nebo procesy začínají v nějakém počátečním časovém bodě, že čas ubíhá dopředu v diskrétních krocích tak, že jednotlivé časové body – momenty – mohou být očíslovány přirozenými čísly. Časový vývoj pak může být znázorněn posloupností přirozených čísel

$$m_0, m_1, m_2, \ldots, m_n, \ldots$$

V tomto případě je časová linka lineární a může být konečná nebo nekonečná. Uvědomme si, že v počátečním momentu není žádná minulost a v každém momentu je jen jedna budoucnost. Takové uspořádání času vyhovuje pro popis sekvenčních procesů. Při popisu distribuovaných procesů se neobejdeme bez složitější struktury – tomuto případu velmi dobře vyhovuje stromová struktura, tedy každý okamžik (časový bod) může mít více bezprostředně následujících časových bodů. Potom má takový moment více budoucností. Pro jednoduchost se omezíme na lineární případ a budeme se ve zbytku kapitoly zabývat lineární temporální logikou (LTL).

8.2.1 Lineární temporální logika

V temporální logice se formule vyhodnocují v množině světů, které odpovídají časovým momentům. Je tedy zřejmé, že při definici sémantiky temporálních operátorů se můžeme opřít o aparát Kripkeho struktur – soustředíme se pouze na Kripkeho struktury, jejichž relace přípustnosti budou odpovídat omezením, která předpokládáme o čase. V případě lineární temporální logiky (LTL) je přípustnost (dosažitelnost) světa s_j ze světa s_i dána uspořádáním $i<j$.

Budeme popisovat lineární temporální logiku, ve které je dosažitelnost určena uspořádáním přirozených čísel. Typickými temporálními operátory jsou:

- $\circ A$, což značí, že *A je pravdivá v následujícím momentu*,
- $\Box A$ vyjadřuje, že *A je pravdivá teď a ve všech následujících momentech*,
- $\Diamond A$ představuje, že *A je pravdivá teď nebo v nějakém následujícím momentu*,
- $A \cup B$, udává, že *A je pravdivá, dokud se nestane pravdivou formule B*.

Pomocí právě zavedených operátorů lze velmi srozumitelným způsobem formálně zapsat chování některých běžných systémů, jak ukážeme na následujících příkladech, kde použité prvotní formule mají zcela přirozenou interpretaci.

Příklad 8.5. Uveďme několik příkladů vztahů platných v běžném životě spolu s jejich formálním popisem, který se bez použití temporálních operátorů neobejde:

a) Vždy platí, že pokud člověk nemá pas, nebo nemá palubní lístek, pak mu v následujícím okamžiku nebude umožněn vstup do letadla

$$\Box((\neg pas \vee \neg palubní_lístek) \rightarrow \circ \neg vstup_do_letadla).$$

Podobně můžeme také vyjádřit tvrzení, že jakmile přednáška začne, bude výklad pokračovat až do okamžiku, kdy je ohlášen konec přednášky

$$začátek_přednášky \rightarrow výklad \cup konec_přednášky;$$

b) Pomocí souboru několika formulí s temporálními operátory můžeme dokonce popsat chování komplexní soustavy. Takové množině formulí se pak říká program pro tento systém. Dobře fungující úřad by mohl být popsán souborem tří formulí, který v dalším výkladu budeme označovat jako program **P**. Vždy platí, že:

1. Jeli požadavek podán, pak bude někdy doručen na správné místo

$$\Box(\textit{požadavek} \rightarrow \Diamond \textit{doručen}),$$

2. Je-li požadavek doručen, pak jeho zpracování bude zahájeno hned v následujícím okamžiku

$$\Box(\textit{doručen} \rightarrow \circ \textit{zpracováván}),$$

3. Zpracovávaný požadavek bude jednou vyřízen (hotov) a pak už zůstane hotov (nebude se znovu otevírat)

$$\Box(\textit{zpracováván} \rightarrow \Diamond\Box \textit{hotov}).$$

Z uvedených formulí se zdá být patrné, že nemůže nastat situace, ve které by systém stále vysílal stejný požadavek, ale ten by nebyl nikdy hotov. V závěru této kapitoly formálně dokážeme, že platí-li všechny tři formule, které tvoří program **P**, pak formule

$$\Box\, \textit{požadavek} \;\&\; \Box\neg \textit{hotov}$$

nemůže platit, čili tato formule je s programem **P** nekonzistentní.

Formule LTL se definují indukcí podobně jako formule modální logiky. I zde bychom mohli využít možnost vzít za základ jazyka jen dvě booleovské spojky, např. $\neg$ a &, a nadefinovat s jejich pomocí ostatní výrokové spojky stejně, jako se to dělá ve výrokové logice. My však výklad zjednodušíme tím, že budeme rovnou pracovat se všemi standardními booleovskými spojkami. Mějme tedy množinu prvotních formulí Φ. Formule jazyka L_Φ výrokové temporální logiky jsou pak pouze formule, které vznikly pomocí jednoho z těchto postupů:

- Každá prvotní formule $p \in \Phi$ je formulí.
- Jsou-li A, B formule, potom následující konstrukty jsou také formulemi:
  $\neg A$, $A \,\&\, B$, $A \vee B$ a $A \rightarrow B$,
  $\circ A$ (příště),
  $\Box A$ (vždy),
  $\Diamond A$ (někdy),
  $A \cup B$ (do té doby než).

8.2.2 Sémantika pro lineární temporální logiku (LTL)

Nechť Φ je množina výrokových konstant, pak *lineární temporální* (nebo *Kripkeho*) *struktura* pro Φ je nekonečná posloupnost pravdivostních ohodnocení

$$\boldsymbol{S} = (\eta_0, \eta_1, \eta_2, \ldots), \text{ kde } \eta_i : \Phi \rightarrow \{\text{false}, \text{true}\}.$$

Prvky této posloupnosti nazýváme *stavy* a jejich číselný index vyjadřuje plynutí času: stav η_{n+1} je bezprostředním následníkem stavu η_n. Říkáme, že η_0 je *počáteční stav* $\boldsymbol{S}$ v časovém bodě 0 a že stav η_{n+1} je stav bezprostředně následující po stavu η_n.

Stavy jsou tedy pravdivostní ohodnocení množiny výrokových konstant Φ a popisují 'stav světa' v časových okamžicích (časových bodech) $m_0, m_1, m_2, m_3, \ldots, m_n, m_{n+1}, \ldots$

Pro temporální strukturu $\boldsymbol{S}$, index i a formuli A *induktivně* definujeme pravdivostní hodnotu $S_i(A)$ formule A v časovém bodě m_i takto:

$S_i(p) = \eta_i(p)$,	právě když	$p \in \Phi$,
$S_i(\neg A) = \text{true}$,	právě když	$S_i(A) = \text{false}$,
$S_i(A \,\&\, B) = \text{true}$,	právě když	$S_i(A) = \text{true}$ a $S_i(B) = \text{true}$,

$S_i(A \vee B)$ = true, právě když $S_i(A)$ = true nebo $S_i(B)$ = true,
$S_i(A \rightarrow B)$ = true, právě když $S_i(A)$ = false nebo $S_i(B)$ = true.
(ekvivalentním vyjádřením je právě když $S_i(A)$ = true implikuje $S_i(B)$ = true).
Pro operátory

$S_i(\circ A) = S_{i+1}(A)$,			
$S_i(\Box A)$ = true,	právě když	$S_j(A)$ = true	pro každé $j \geq i$,
$S_i(\Diamond A)$ = true,	právě když	$S_j(A)$ = true	pro nějaké $j \geq i$,
$S_i(A \cup B)$ = true,	právě když	$S_j(B)$ = true	pro nějaké $j, j > i$ a $S_k(A)$ = true pro každé k ležící mezi i a j $(i \leq k < j)$.

Pravdivostní hodnotu můžeme obvyklým způsobem definovat i pro další odvozené spojky, např.

$$S_i(A \leftrightarrow B) = \text{true, právě když } S_i(A) = S_i(B).$$

Není těžké najít nutnou a postačující podmínku pro to, aby v nějakém konkrétním stavu η_i platila formule $(\neg \Box \neg A)$. Tato formule platí ve stavu η_i právě tehdy, pokud existuje $j \geq i$ takové, že A platí v η_j. Předveďme stručně postup, kterým se o tom přesvědčíme.

$S_i(\neg \Box \neg A)$ = true,	právě když	$S_i(\Box \neg A)$ = false,	
$S_i(\Box \neg A)$ = false,	právě když	$S_j(\neg A)$ = false	pro nějaké $j \geq i$,
$S_j(\neg A)$ = false,	právě když	$S_j(A)$ = true,	tedy
$S_i(\Box \neg A)$ = false,	právě když	$S_j(A)$ = true	pro nějaké $j \geq i$
$S_i(\neg \Box \neg A)$ = true,	právě když	$S_j(A)$ = true	pro nějaké $j \geq i$.

Dospěli jsme k podmínce, která charakterizuje operátor $\Diamond$. Tak jsme ověřili, že formule $(\Diamond A)$ a $(\neg \Box \neg A)$ jsou vzájemně ekvivalentní.

Definice. Zaveďme validitu a sémantický důsledek. Nechť A je formule jazyka L_Φ logiky LTL(Φ). Říkáme, že

a) A platí v temporální struktuře $\boldsymbol{S}$ pro Φ, (A je *splněna* ve struktuře $\boldsymbol{S}$) a píšeme $\boldsymbol{S} \models A$ nebo $\models_S A$, jestliže $S_i(A)$ = true pro všechna i.

b) A je splnitelná v temporální struktuře $\boldsymbol{S}$ pro Φ, jestliže existuje nějaký okamžik (stav) i takový, že $S_i(A)$ = true.

Pokud je formule A splněna ve všech lineárních temporálních strukturách $\boldsymbol{S}$ pro Φ, označujeme ji jako pravdivou (validní) *formuli* LTL nebo krátce jen jako validní formuli, pokud je z kontextu zřejmé, že pracujeme s LTL.

Nechť $\boldsymbol{T}$ je množina formulí jazyka L_Φ. Říkáme, že struktura $\boldsymbol{S}$ je *modelem množiny formulí* $\boldsymbol{T}$, jestliže všechny formule B z $\boldsymbol{T}$ jsou ve struktuře $\boldsymbol{S}$ splněny (platí pro ně $\boldsymbol{S} \models B$). Říkáme, že A je (*sémantický*) *důsledek* $\boldsymbol{T}$ a píšeme $\boldsymbol{T} \models A$, jestliže A je splněna v každém modelu $\boldsymbol{S}$ množiny $\boldsymbol{T}$. Je-li $\boldsymbol{T}$ prázdná množina, píšeme $\models A$, v tomto případě je A splněna ve všech temporálních strukturách pro jazyk L_Φ.

VĚTA 8.4 (o korektnosti pravidla modus ponens). Nechť $\boldsymbol{S}$ je lineární temporální struktura a i je přirozené číslo, nechť $S_i(A)$ = true a $S_i(A \rightarrow B)$ = true, potom $S_i(B)$ = true.
Důkaz. V libovolném časovém bodě je pravdivostní ohodnocení implikace definováno zcela klasickým způsobem, tj. $S_i(A \rightarrow B)$ = true právě tehdy, když $S_i(A)$ = false nebo $S_i(B)$ = true. Z předpokladu $S_i(A)$ = true je tedy zřejmé, že musí platit $S_i(B)$ = true.

Z věty 8.4 plynou důsledky:

- Je-li $\boldsymbol{T}$ množina formulí, pro kterou platí, že $\boldsymbol{T} \models A$ a $\boldsymbol{T} \models A \to B$, potom $\boldsymbol{T} \models B$.
- Snadno se nahlédne, že platí: je-li $\boldsymbol{T} \models A$ a $\boldsymbol{T}$ je množina validních formulí, potom $\models A$.

Příklad 8.6. Přesvědčíme se o validitě několika následujících formulí LTL.

a) Formule $(\neg\circ A \leftrightarrow \circ\neg A)$ je splněna ve všech strukturách pro LTL.

Stačí ukázat, že $S_i(\neg\circ A) = S_i(\circ\neg A)$ pro libovolnou strukturu $\boldsymbol{S}$ a kterýkoliv časový bod i. To je zřejmé z následujících úvah:

$S_i(\neg\circ A) = \text{true}$, právě když $S_i(\circ A) = \text{false}$,
$S_i(\circ A) = \text{false}$, právě když $S_{i+1}(A) = \text{false}$,
$S_{i+1}(A) = \text{false}$, právě když $S_{i+1}(\neg A) = \text{true}$,
$S_{i+1}(\neg A) = \text{true}$, právě když $S_i(\circ\neg A) = \text{true}$.

b) Podobně ukážeme, že i formule $\circ(A \to B) \leftrightarrow (\circ A \to \circ B)$ je validní:

$K_i(\circ(A \to B)) = \text{true}$, právě když $K_{i+1}(A \to B) = \text{true}$,
právě když $K_{i+1}(\neg A) = \text{true}$ nebo $K_{i+1}(B) = \text{true}$,
právě když $K_{i+1}(A) = \text{false}$ nebo $K_{i+1}(B) = \text{true}$,
právě když $K_i(\circ A) = \text{false}$ nebo $K_i(\circ B) = \text{true}$,
právě když $K_i(\neg\circ A) = \text{true}$ nebo $K_i(\circ B) = \text{true}$,
právě když $K_i(\circ A \to \circ B) = \text{true}$.

Právě jsme ověřili, že operátor *next* se distribuuje přes implikaci. Jde o jednoduchý důsledek toho, že pravdivost formule $\circ A$ závisí na pravdivosti A v jediném časovém bodě. Stačí využít obvyklých vlastností výrokových spojek, abychom dokázali, že operátor *next* se distribuuje i přes ostatní výrokové spojky.

c) Ukážeme, že formule $\Box A \to A$ je splněna ve všech strukturách lineární temporální logiky. Buď i libovolný časový bod struktury $\boldsymbol{S}$. Ukážeme, že $S_i(\Box A \to A) = \text{true}$. Platí totiž

$S_i(\Box A \to A) = \text{true}$, právě když $S_i(\Box A) = \text{false}$ nebo $S_i(A) = \text{true}$.

Rozlišíme dva případy:

- Je-li $S_i(\Box A) = \text{false}$, je tvrzení zřejmé z definic pravdivosti pro výrokové spojky.
- Nechť $S_i(\Box A) = \text{true}$, pak definice splňování modality $\Box$ požaduje, aby A platilo v časovém bodě i a ve všech bodech následujících. Tedy platnost tvrzení $S_i(A) = \text{true}$ i $S_i(\Box A \to A) = \text{true}$ je zcela zřejmým důsledkem této definice.

d) Snadno se ověří, že $\Box A \to (A \wedge \circ\Box A)$ je také validní formulí.

Důkaz rozložíme do dvou kroků. Nejprve se přesvědčíme, že je validní jednodušší formule $\Box A \to \circ\Box A$. Výchozí formuli z bodu d) pak získáme jako výrokový důsledek této formule a formule z bodu c). Stačí si totiž uvědomit, že v každém časovém bodě platí výroková tautologie $((\alpha\to\beta) \wedge (\alpha\to\gamma)) \to (\alpha\to (\beta\wedge\gamma))$ a použít její instanci s předpoklady tvořenými validními formulemi $(\Box A \to A)$ a $(\Box A \to \circ\Box A)$ a dvakrát aplikovat pravidlo modus ponens (viz věta 8.4). Musíme tedy ukázat, že formule $\Box A \to \circ\Box A$ je validní

$$S_i(\Box A \to \circ\Box A) = \text{true}, \text{ právě když } S_i(\Box A) = \text{true implikuje } S_i(\circ\Box A) = \text{true}.$$

Přitom z definice $S_i(\Box A) = \text{true}$ vyplývá, že $S_i(A) = \text{true}$ a $S_{i+1}(\Box A) = \text{true}$, a tedy musí platit

$$S_i(\circ\Box A) = \text{true}.$$

Dokonce se dá dokázat, že i obrácená implikace $(A \,\&\, \circ\Box A) \to \Box A$ je validní.
Jak uvidíme později, formule ověřené v příkladu 8.6 velmi dobře charakterizují LTL, a proto se s nimi setkáme, až budeme budovat formální systém pro LTL (jsou použity při její axiomatizaci).

VĚTA 8.5. Nechť $\boldsymbol{T}$ je množina formulí jazyka LTL a A formule. Je-li $\boldsymbol{T} \models A$, potom platí také $\boldsymbol{T} \models \circ A$ a $\boldsymbol{T} \models \Box A$. Speciálně $A \models \circ A$ a $A \models \Box A$.

Důkaz. Nechť $\boldsymbol{S}$ je libovolná temporální struktura, která je modelem $\boldsymbol{T}$, a nechť i, j jsou přirozená čísla. Podle předpokladu platí, že $S_i(A) = \text{true}$ pro všechna i. Speciálně tedy také pro všechny časové body j musí platit $S_j(A) = \text{true}$ a $S_{j+1}(A) = \text{true}$, a tedy $S_j(\circ A) = \text{true}$. Podobně $S_j(A) = \text{true}$ pro všechna $j, j \geq i$, a tedy $S_j(\Box A) = \text{true}$. To znamená, že $\boldsymbol{T} \models \circ A$ a $\boldsymbol{T} \models \Box A$.

Zajímavější je intuitivně zřejmé tvrzení o indukci uvedené ve větě 8.6.

VĚTA 8.6. Je-li $\boldsymbol{T} \models A \to B$ a $\boldsymbol{T} \models A \to \circ A$, potom $\boldsymbol{T} \models A \to \Box B$.

Příslušný důkaz si může čtenář doplnit sám. Stačí dovést ke sporu předpoklad, že v nějaké struktuře $\boldsymbol{S}$ pro $\boldsymbol{T}$ existuje stav j takový, že $S_j(B) = \text{false}$.

Věty 8.5 a 8.6 formulují odvozovací pravidla *next* a *indukce*, která budou použita při konstrukci formálního systému LTL, a dokazují jejich korektnost: tato pravidla dovolují odvodit z validních formulí opět validní formule. Při budování formálního systému pro LTL nás bude jistě zajímat i to, zda platí nějaká obdoba klasické věty o dedukci. Odpovědí na tuto otázku je věta 8.7.

VĚTA 8.7. $\boldsymbol{T} \cup \{A\} \models B$, právě když $\boldsymbol{T} \models \Box A \to B$.

Důkaz. Nechť $\boldsymbol{T} \models \Box A \to B$ a $\boldsymbol{S}$ je libovolná struktura pro $\boldsymbol{T} \cup \{A\}$. Podle věty 8.5 je $\boldsymbol{S}$ strukturou i pro $\boldsymbol{T} \cup \{\Box A\}$, a tedy použitím modus ponens dostáváme $\boldsymbol{S} \models B$.

Naopak, nechť $\boldsymbol{T} \cup \{A\} \models B$ a $\boldsymbol{S}$ je struktura pro $\boldsymbol{T}$. Pokud $\boldsymbol{S} \models A$, pak $\boldsymbol{S} \models B$ i $\boldsymbol{S} \models \Box A \to B$ platí. Není-li $\boldsymbol{S} \models A$, pak je množina $M = \{i: S_i \not\models A\} \neq \emptyset$. Pokud M není konečná, musí pro všechna j platit $S_j \not\models \Box A$, a tedy $S_j \models \Box A \to B$. Je-li M konečná, označíme i_0 její největší prvek. Pak pro každé $j > i_0$ platí $S_j \models A$ a máme tedy strukturu $\boldsymbol{T} \cup \{A\}$, kde podle předpokladu platí B, a tedy i $S_j \models \Box A \to B$. Naopak pro $j \leq i_0$ je $S_j \not\models \Box A$, a tedy $S_j \models \Box A \to B$.

VĚTA 8.8. Je-li $\boldsymbol{T} \models A \to B$ potom $\boldsymbol{T} \cup \{A\} \models B$.

Důkaz. Nechť $\boldsymbol{S}$ je libovolná struktura pro $\boldsymbol{T} \cup \{A\}$ a nechť i je libovolný její časový bod. Je zřejmé, že $S_i(A) = \text{true}$ a z předpokladu o $\boldsymbol{T}$ plyne, že v bodě i musí platit i $A \to B$. Platnost $S_i(B) = \text{true}$ je pak už triviálním důsledkem pravidla Modus ponens.

Stojí za povšimnutí, že implikace opačná než ta uvedená ve větě 8.8 v LTL neplatí. To znamená, že na rozdíl od klasické výrokové logiky v LTL platí jen neúplná obdoba klasické „věty o dedukci". Abychom se o tom přesvědčili, stačí uvažovat případ, kdy $\boldsymbol{T}$ je prázdná množina. Podle věty 8.5 platí $p \models \Box p$ pro libovolnou prvotní formuli. Ovšem implikace $p \to \Box p$ nemusí být validní formulí, není totiž pravdivá v žádné temporální struktuře $\boldsymbol{S}$, kde pro nějaké i a $j > i$ platí $S_i(p) = \text{true}$ a $S_j(p) = \text{false}$.

VĚTA 8.9. $\models A$, právě když $\neg A$ není splnitelná.
Důkaz získáme prostým přepisem definic:

$\models A$, právě když $S_i(A)$ = true pro libovolnou strukturu $\boldsymbol{S}$ a libovolné i,

právě když $S_i(\neg A)$ = false pro libovolnou strukturu $\boldsymbol{S}$ a libovolné i,

právě když neexistuje struktura $\boldsymbol{S}$ a její stav j takový, že $S_j(\neg A)$ = true.

Tvrzení na poslední řádce odpovídá definici pojmu „Formule $\neg A$ není splnitelná". Stejně jako v klasické logice je i v LTL validita „duálním" pojmem ke splnitelnosti.

Příklad 8.7. (a) Velmi užitečné jsou i další validní formule ve tvaru ekvivalencí, které umožňují zjednodušovat formule temporální logiky například tím, že dovolují přesouvat negaci co nejblíže k prvotním formulím

$$\models \Box\neg A \leftrightarrow \neg \Diamond A, \qquad \models \Diamond\neg A \leftrightarrow \neg\Box A, \qquad \models \circ\neg A \leftrightarrow \neg\circ A.$$

Důkaz validnosti těchto formulí lze podat podobným způsobem jako je ten, který jsme uplatnili v předchozích příkladech, např. v příkladu 8.6. Složitější vztahy pak lze z těchto výchozích vztahů odvodit. Platí například

$$\models \Box\Diamond\neg A \leftrightarrow \neg\Diamond\Box A, \quad \models \Diamond\Box\neg A \leftrightarrow \neg\Box\Diamond A.$$

(b) Naopak následující validní implikace nelze zesílit na ekvivalence

$\models A \rightarrow \Diamond A$, $\qquad \models \Box A \rightarrow A$,
$\models \circ A \rightarrow \Diamond A$, $\qquad \models \Box A \rightarrow \circ A$,
$\models \Box A \rightarrow \Diamond A$, $\qquad \models \Box A \rightarrow \circ\Box A$,
$\models A \cup B \rightarrow \Diamond A$, $\qquad \models \Diamond\, \Box A \rightarrow \Box\Diamond A$.

(c) Zajímavé je vícenásobné použití téhož temporálního operátoru, např. operátory $\Diamond$, $\Box$, $\Box\Diamond$, $\Box$ jsou idempotentní, tj.

$\models \Diamond\Diamond A \leftrightarrow \Diamond A$, $\qquad \models \Box\Box A \leftrightarrow \Box A$,
$\models \Diamond\Box\Diamond\Box A \leftrightarrow \Diamond\Box A$, $\qquad \models \Box\Diamond\Box\Diamond A \leftrightarrow \Box\Diamond A$.

Naopak operátor *next* není idempotentní, neboť formule $\circ\circ A \leftrightarrow \circ A$ docela určitě není validní.

(d) Povšimneme si, že uvedené operátory jsou monotónní v každém argumentu.

$\models \Box(A \rightarrow B) \rightarrow (\Box A \rightarrow \Box B)$, $\qquad \models \Box(A \rightarrow B) \rightarrow (\Diamond A \rightarrow \Diamond B)$,
$\models \Box(A \rightarrow B) \rightarrow (\circ A \rightarrow \circ B)$,
$\models \Box(A \rightarrow B) \rightarrow (\Diamond\Box A \rightarrow \Diamond\Box B)$, $\qquad \models \Box(A \rightarrow B) \rightarrow (\Diamond\Box A \rightarrow \Diamond\Box B)$,
$\models \Box(A \rightarrow B) \rightarrow ((A \cup C) \rightarrow (B \cup C))$,
$\models \Box(A \rightarrow B) \rightarrow ((C \cup A) \rightarrow (C \cup B))$.

8.2.3 Dokazatelnost v LTL

Formální systém Lineární temporální logiky je tvořen axiomy, které představují typické struktury validních formulí, a odvozovacími pravidly. Pro libovolné formule A, B jsou následující formule axiomy LTL:

(Taut) tvoří instance všech tautologií výrokové logiky,
(LTL 1) vyjadřuje $\neg\circ A \leftrightarrow \circ\neg A$,
(LTL 2) představuje $\circ(A \to B) \leftrightarrow (\circ A \to \circ B)$,
(LTL 3) značí $\Box A \leftrightarrow (A \,\&\, \circ\Box A)$.

Odvozovací pravidla LTL popisují formální postup, jak z jedné nebo dvou validních formulí získat další validní formuli:

(Modus ponens) umožňuje z formulí A a $A \to B$ odvodit B,
(Next) představuje postup odvození $\circ A$ z formule A,
(Indukce) dovoluje z formulí $A \to B$ a $A \to \circ A$ odvodit $A \to \Box$.

V předchozím oddílu o sémantice jsme ověřili, že axiomy (Taut) a (LTL1) až (LTL3) jsou validní formule LTL, a věty 8.5 a 8.6 dokazují, že aplikací odvozovacích pravidel tvoříme z validních formulí opět validní formule.

VĚTA 8.10 (jednoduché důsledky axiomů). Pro libovolné formule A, B jsou uvedená tvrzení dokazatelná (značka $\vdash$ se používá pro „je dokazatelné“) ve formálním systému LTL:

a) $\vdash \Box A \to A$,
b) $\vdash \Box A \to \circ\Box A$.

Následující odvozovací pravidla představují varianty pravidla (Indukce):

c) $A \to \circ A \vdash A \to \Box A$ (ind1),
d) $A \to B, B \to \circ B \vdash A \to \Box$ (ind2).

Další odvozovací pravidlo je obdobou pravidla (Next) pro $\Box$:

e) $A \vdash \Box A$ (alw),
f) $A \to \circ B \vdash A \to \Diamond B$ (som).

Důkaz. Implikace a) a b) plynou bezprostředně z axiomu LTL 3. Vztah c) formuluje nové indukční pravidlo, jehož platnost ověříme tím, že zkonstruujeme důkaz v příslušném formálním systému. Budeme tedy tvořit posloupnost formulí doplněnou (vždy v pravém sloupci) argumentem, o který se platnost uváděného tvrzení opírá:

1.	$A \to \circ A$	předpoklad,
2.	$\vdash A \to A$	(Taut),
3.	$A \to \circ A \vdash A \to \Box A$	pravidlo (Indukce) na formule 1 a 2.

Podobě lze zkonstruovat i důkaz pro pravidlo (alw):

1.	A	předpoklad,
2.	$A \vdash \circ A$	pravidlo (Next) aplikované na formuli 1,
3.	$\vdash \circ A \to (A \to \circ A)$	(Taut),
4.	$A \vdash A \to \circ A$	Modus ponens na formule 3 a 2,
5.	$A \vdash A \to \Box A$	pravidlo (ind1) aplikované na formuli 4.,
6.	$A \vdash \Box A$	Modus ponens na formule 1 a 5.

Důkaz posledního tvrzení (som) necháváme na čtenáři, neboť se příliš neliší od předchozího postupu.

VĚTA 8.11 (o dedukci pro LTL). Nechť A, B jsou formule a T je množina formulí. Potom platí

$$T \cup \{A\} \vdash B, \text{ právě když } T \vdash \Box A \to B.$$

Důkaz. Postupujeme indukcí podle délky odvození. Začneme implikací zleva doprava.

a) Je-li B axiom LTL nebo B je prvkem T, potom $T \vdash B$, a tedy pomocí instancí axiomů výrokové logiky (Taut) odvodíme $T \vdash \Box A \to B$.

b) Nechť je B totožná s formulí A. Pak je naším cílem dokázat, že $T \vdash \Box A \to A$. Toto tvrzení jistě platí, viz bod a) věty 8.9.

c) Je-li B odvozena pravidlem (Modus ponens) z formulí C a $C \to B$. Potom z indukčních předpokladů vyplývá

$$T \vdash \Box A \to C \text{ a } T \vdash \Box A \to (C \to B),$$

odkud pomocí (Taut) a (Modus ponens) odvodíme

$$T \vdash \Box A \to B.$$

d) Je-li B tvaru $\circ C$ a je odvozena z C pomocí pravidla (Next), z indukčního předpokladu dostáváme

$$T \vdash \Box A \to C$$

a znovu pomocí (Next)

$$T \vdash \circ(\Box A \to C).$$

Potom podle LTL2 a (Modus ponens) je $T \vdash \circ\Box A \to \circ C$. Dále víme, že $\Box A \to \circ\Box A$ je důsledkem LTL3. Složením těchto dvou implikací dostaneme

$$T \vdash \Box A \to \circ C.$$

e) Pokud B je tvaru $C \to \Box D$ a je odvozena pravidlem (indukce) z dřív dokázaných formulí $C \to D$ a $C \to \circ C$, pak z indukčních předpokladů získáme

$$T \vdash \Box A \to (C \to D),$$
$$T \vdash \Box A \to (C \to \circ C).$$

Cílové tvrzení $T \vdash \Box A \to (C \to \Box D)$ dokážeme ve tvaru $T \vdash (\Box A \,\&\, C) \to \Box D$ pomocí pravidla (Indukce) následujícím postupem. Z obou předpokladů nejprve pomocí (Taut) dostáváme

$$T \vdash (\Box A \,\&\, C) \to D,$$
$$T \vdash (\Box A \,\&\, C) \to \circ C.$$

Využijeme tvrzení $\vdash \Box A \to \circ\Box A$ bodu b) věty 8.10 a složením obou implikací obdržíme

$$T \vdash (\Box A \,\&\, C) \to (\circ\Box A \,\&\, \circ C).$$

Dvojnásobným použitím LTL2 na tautologii $\Box A \to (C \to (\Box A \,\&\, C))$ dostaneme další formuli $\circ\Box A \to (\circ C \to \circ(\Box A \,\&\, C)$, kterou lze dále upravit do tvaru spojujícího oba předchozí

$$\vdash (\circ\Box A \,\&\, \circ C) \to \circ(\Box A \,\&\, C) \qquad \text{LTL2.}$$

Celkem tedy máme

$$T \vdash (\Box A \,\&\, C) \to \circ(\Box A \,\&\, C)$$

a nakonec díky platnosti $T \vdash (\Box A \,\&\, C) \to D$ můžeme aplikovat pravidlo (Indukce) a obdržíme

$$T \vdash (\Box A \,\&\, C) \to \Box D.$$

Tím je dokázána jedna implikace věty o dedukci.

Implikace zprava doleva má důkaz kratší. Předpokládejme, že $T \vdash \Box A \to B$, potom také $T \cup \{A\} \vdash \Box A \to B$, přitom $T \cup \{A\} \vdash \Box$A, a to podle (alw). Stačí použít (modus ponens) a dostaneme

$$T \cup \{A\} \vdash B.$$

Tím je věta o dedukci pro LTL dokázána.

Poznámka. Z věty o dedukci klasické výrokové logiky se stejným způsobem odvodí jen implikace „Je-li $T \vdash A \to B$, potom $T \cup \{A\} \vdash B$. Opačnou implikaci nelze v LTL dokázat.

Příklad 8.8. Následující formule platí v každé struktuře lineární temporální logiky:

a) $\vdash \Box(A \to B) \to (\Box A \to \Box B)$,

b) $\vdash \Diamond(A \to B) \to (\Diamond A \to \Diamond B)$.

Uvedené formule, tak typické pro modální logiky, se dají dokázat pomocí věty o dedukci. Důkaz provedeme pro první z nich, která upozorňuje, že operátor $\Box$ se distribuuje přes implikaci:

1. $(A \to B), A \vdash B$ (Modus ponens).
2. $(A \to B), A \vdash \Box B$ (pravidlo alw aplikované na formuli1.),
3. $(A \to B) \vdash \Box A \to \Box B$ (věta o dedukci pro formuli 2.),
4. $\vdash \Box(A \to B) \to (\Box A \to \Box B)$ (věta o dedukci aplikovaná na formuli 3.).

Nyní dokončíme část b) z příkladu 8.5. Jsme konečně připraveni ověřit, že zde uvedené tvrzení skutečně platí. Můžeme použít vybudovaný formální systém, abychom se ujistili, že uvedený program zaručuje, že každý požadavek bude (někdy) zpracováván a výsledek bude hotov. Připomeňme si výchozí program:

1. $\Box(požadavek \to \Diamond doručen)$,
2. $\Box(doručen \to \circ zpracováván)$,
3. $\Box(zpracováván \to \Diamond\Box hotov)$.

Nejprve postupujeme pozitivně a použijeme tato obecně platná tvrzení:

a) $\Box A \to \Diamond A$,
b) $\circ A \to \Diamond A$,
c) $\Box(A \to B) \to (\Box A \to \Box B)$,
d) $\Diamond(A \to B) \to (\Diamond A \to \Diamond B)$,
e) $\Diamond\Diamond A \leftrightarrow \Diamond A$.

Potom můžeme dovozovat:
podle 1., c) a (mp) je
f) $\Box požadavek \rightarrow \Box\Diamond$ *doručen*,
podle a) máme
g) $\Box\Diamond$ *doručen* $\rightarrow \Diamond\Diamond$*doručen*,
$\Box$(*doručen* $\rightarrow \circ$*zpracováván*) $\rightarrow \Diamond$(*doručen* $\rightarrow \circ$*zpracováván*),
podle d) dostaneme
$\Diamond$(*doručen* $\rightarrow \circ$*zpracováván*) $\rightarrow$ ($\Diamond$*doručen* $\rightarrow \Diamond\circ$*zpracováván*),
z (Taut) a b) získáme
($\Diamond$*doručen* $\rightarrow \Diamond\circ$*zpracováván*) $\rightarrow$ ($\Diamond$*doručen* $\rightarrow \Diamond\Diamond$*zpracováván*),
podle e) dostaneme
$\Diamond\Diamond$*zpracováván* $\rightarrow \Diamond$*zpracováván*.
Nyní stačí využít 2, několikanásobně aplikovat Modus ponens a pomocí (Taut) obdržíme
h) $\Diamond$*doručen* $\rightarrow \Diamond$*zpracováván*.
Podle 3, a), c) a (Taut) máme
$\Diamond$*zpracováván* $\rightarrow \Diamond\Diamond\Box$*hotov*,
čehož lze díky e) zjednodušit na
i) $\Diamond$*zpracováván* $\rightarrow \Diamond\Box$*hotov*,
Nakonec postupným spojením tvrzení f) až i) máme
$\Box$*požadavek* $\rightarrow \Diamond$*zpracováván*, a také $\Box$*požadavek* $\rightarrow \Diamond\Box$*hotov*.
Ověřili jsme, že každý požadavek bude někdy vyřízen.
Můžeme také postupovat negativně a dokázat, že tvrzení
$\Box$*požadavek* & $\Box\ \neg$*hotov*
vede ke sporu. Vyjdeme-li z (Taut), popř. u druhé formule z (Taut) a definice $\Diamond$, získáme
$\Box$*požadavek* & $\Box\neg$*hotov* $\rightarrow \Box\neg$*hotov*,
$\Box\neg$ *hotov* $\leftrightarrow \neg\Diamond$ *hotov*.
Dostaneme tedy
($\Box$*požadavek* & $\Box\neg$*hotov*) $\rightarrow \neg\Diamond$*hotov*,
a to je ve sporu s dříve dokázaným tvrzením. Stačí si totiž uvědomit, že platí
$\Diamond\Box$*hotov* $\rightarrow \Diamond$*hotov*.
Ověřili jsme, že program vyřizuje požadavky korektně.

VĚTA 8.12 (o korektnosti pro LTL). Nechť T je libovolná množina formulí. Je-li formule A dokazatelná z formulí T, potom A je validní formule, tedy

$$\vdash_{LTL} A \text{ implikuje } \models A.$$

Důkaz se provádí podobně jako u věty o dedukci indukcí podle délky odvození formule A v LTL. Stačí si uvědomit, že axiomy LTL jsou validní formule a odvozovací pravidla z validních formulí odvozují validní formule. Porobnosti opět necháme na čtenáři.

8.3 Závěrečné poznámky

Modální logice je v současné době věnována značná pozornost nejen proto, že nabízí prostředky pro formalizaci uvažování v multiagentních systémech. Vychází proto řada publikací, které se věnují jak jejím teoretickým, tak praktickým aspektům. Již v textu kapitoly jsme několikrát uvedli skvělou knihu (Fagin a spol., 1995). Důkladný přehled teoretických zákla-

dů nabízí například publikace (Blackburn a spol., 2002). Naopak jako populární úvod a motivace ke studiu této disciplíny určitě dobře poslouží poutavá sbírka příkladů *Navěky nerozhodnuto* (Smullyan, 2003), jejíž původní anglické vydání vyšlo v roce 1987. V doslovu k této knížce (Hromek, 2003) nabízí překladatel zajímavý přehled o historickém kontextu i řadu dalších doporučení na studijní literaturu.

Formální základy k současnému pojetí temporální logiky položil Arthur Prior ve své knize *Past, Present and Future* (Prior, 1967). Koncem sedmdesátých let ukázal Amir Pnueli na možnosti, které temporální logika nabízí pro specifikaci a verifikaci počítačových systémů (Pnuelli, 1977) a od té doby se s ní setkáváme jako se standardním nástrojem pro dokazování vlastností počítačových systémů (Manna, Pnuelli, 1995). Velmi přehledný pohled na současné dění v oblasti temporální logiky nabízí například učební text (Goranko, 2009) připravený pro první podzimní Ph.D. školu modální logiky, kterou pořádala IT Universita v Kodani v roce 2009.

Literatura

Blackburn B. P., de Rijke M. a Venema Y.: *Modal logics.* Cambridge University Press, 2002.

Fagin R., Halpern J. Y., Mores Y., Vardi M. Y.: *Reasoning about Knowledge.* MIT Press, 1995 a 2003.

Goranko V.: *Introduction to Temporal logics for Specification and Verification* (viz http://hylocore.ruc.dk/m4m6school.html).

Hintikka J.: *Knowledge and Belief – An Introduction to the Logic of the Two Notions.* Cornell University Press, 1962.

Hromek P.: O moderní logice, metamatematice a logice dokazatelnosti. In: Smullyan R.: *Navěky nerozhodnuto*. Academia, Praha 2003, s. 280–304.

Kripke S. A.: Semantical considerations on modal logic, *Acta Philosophica Fennica 24*, 1963, s. 83–94.

Lewis C. I.: *A survey of symbolic logic*. University of California Press, Berkeley 1918.

Manna Z., Pnuelli A.: *Temporal verification of reactive systems: safety*. Springer Verlag, New York 1995.

Pnuelli A.: The temporal logic of programs. In: *Proc. 18th Symp. Foundations of Computer Science*. 1977, s. 46–57.

Prior A.: *Past, Present and Future*. Oxford University Press, 1967.

Smullyan R.: *Navěky nerozhodnuto*. Academia, Praha 2003.

Štěpánková O., Štěpánek P.: Řešení úloh a dokazování vět. In: Mařík V., Štěpánková O., Lažanský J. (eds): *Umělá inteligence 1*. Academia, Praha 1993, s. 67–98.

Urban J., Vyskočil J., Štěpánek P.: Automatické uvažování. In: Mařík V., Štěpánková O., Lažanský J. (eds): *Umělá inteligence 6*. Academia, Praha 2013, s.249–279.

9 Využívání znalostí pro získávání znalostí

Získávání znalostí z dat (data mining) lze dle publikací (Boulicaut, Jeudy, 2005), (Dehaspe, De Raedt, 1997) a (Mannila, 1995) obecně definovat jako sestavení množiny

$$T = \{\phi \in \mathcal{L} \mid \mathrm{q}(D, \phi) \text{ je pravda}\}, \tag{9.1}$$

kde D jsou vstupní data, $\mathcal{L}$ značí nějaký formální *jazyk*, tedy spočetnou množinu formulí, a predikát q vyhodnocuje kvalitu formule $\phi \in \mathcal{L}$ vzhledem k D. Například $\mathcal{L}$ může být jazykem *asociačních pravidel* (Berka, 2003) nad znaky zastupujícími položky zboží a D databáze transakcí. V obvyklém případě by potom $\mathrm{q}(D, \phi)$ bylo vyhodnoceno jako pravdivé právě tehdy, pokud pravidlo ϕ dosáhne požadovaných hodnot *podpory* a *spolehlivosti* na datech D.

Množina T reprezentuje znalost získanou z D. Formulím $\phi \in T$ také říkáme *vzory*. Množina T je v pracích (Mitchell, 1980), (Dehaspe, De Raedt, 1997) nazývána *teorií*, ale je třeba mít na paměti, že v tomto obecném pojetí získávání znalostí její prvky nemusejí být větami formální logiky. Mohou to být třeba aritmetické vztahy či jiné struktury, jejichž význam vzhledem k datům dokážeme interpretovat a jejichž soulad s daty umíme vyhodnotit pomocí q.

Informační hodnota či užitečnost získané znalosti nutně neroste s velikostí výsledné množiny T. V mnoha případech se jeví naopak problémem, že T je příliš velká. Tak tomu bývá třeba v již zmíněném příkladě těžení asociačních pravidel. V některých úlohách data miningu je dokonce obvyklým cílem najít pouze jediný vzor $\hat{\phi}$, jehož kvalita vzhledem k datům D je co největší. Tento případ formalizujme jako hledání $T = \{\hat{\phi}\}$, kde

$$\hat{\phi} = \arg\max_{\phi \in \mathcal{L}} q(D, \phi), \tag{9.2}$$

kde q je tentokrát zobrazením do oboru nezáporných reálných čísel vystihujícím kvalitu ϕ vzhledem k D, význam ostatních symbolů zůstává stejný.

Z obou uvedených rovnic je patrné, že k sestrojení T potřebujeme $\mathcal{L}$. Přitom samotný jazyk $\mathcal{L}$ formálně odpovídá pojmu znalosti vymezenému rovnicí (9.1), neboť predikát q lze stanovit tak, že $T = \mathcal{L}$. Jinými slovy, k získávání (aposteriorní) znalosti T nutně potřebujeme nějakou vstupní (apriorní) znalost $\mathcal{L}$ definující, „kde" výstupní znalost hledat. Tedy název kapitoly vystihuje nikoliv možnost, ale nutnou součást data miningu.

9.1 Zaujetí algoritmu

Protože dataminingový algoritmus při konstrukci množiny T bere v úvahu pouze prvky jazyka $\mathcal{L}$, je $\mathcal{L}$ nazýván *zaujetím* (bias) algoritmu. Zaujetí je nutnou součástí každého dataminingového algoritmu, neboť ten bude vždy mít pouze konečné výrazové prostředky definující (třebaže implicitně) jazyk $\mathcal{L}$. Je také zřejmé, že čím je zaujetí silnější ($\mathcal{L}$ menší), tím je větší riziko, že některé zajímavé vzory nebude schopen algoritmus vyjádřit, a tedy „objevit".

Silné zaujetí má ale své výhody. Uvažujme například učení binárních klasifikátorů, což je speciální případ těžení vzorů. Zde uvažujeme nějakou (pro jednoduchost konečnou) množinu objektů X, přičemž data D jsou její podmnožinou. Předpokládáme dále, že každý prvek $x \in X$ má *kladnou* nebo *zápornou* třídu, ale třídy jsou nám známy pouze pro prvky dat $x \in D \subseteq X$. Máme tedy k dispozici množinu D_+ zahrnující právě všechny kladné prvky D.

Vzory objevené v D by nám měly posloužit k určování tříd i ostatních objektů $x \in X \setminus D$. Každý vzor $\phi \in \mathcal{L}$ má tedy definovat nějakou podmnožinu $\phi(X) \subseteq X$, o jejíchž prvcích řekneme, že jsou vzorem ϕ klasifikovány jako kladné. Při vyhledávání vzorů pak predikát $q(D, \phi)$ z rovnice (9.1) bude pravdivý právě tehdy, když ϕ bude „konzistentní" se známou klasifikací na D, tedy když $\phi(X) \cap D = D_+$. Získanou množinu vzorů T použijeme pro klasifikaci tak, že její prvky necháme hlasovat, tedy $x \in X$ bude klasifikován kladně právě tehdy, když $|\{\phi \in T \mid x \in \phi(X)\}| > |T|/2$. Při opačné nerovnosti bude klasifikace záporná, při rovnosti nemůžeme rozhodnout.

Uvažujme nyní o volbě zaujetí $\mathcal{L}$ v této úloze. Nejde nám teď o konkrétní syntaktickou formu vzorů, mohou jí být třeba logická pravidla, neuronové sítě, algebraické mnohočleny atd. Předpokládejme pouze, že pro každou podmnožinu $X' \subseteq X$ existuje vzor $\phi \in \mathcal{L}$ takový, že $\phi(X) = X'$. Jakmile je tato podmínka splněna, pozorujeme dvě věci. Zaprvé pro jakákoliv data D z X bude množina nalezených vzorů T neprázdná, tedy vždy najdeme alespoň jeden konzistentní vzor. Zadruhé nemá smysl jazyk $\mathcal{L}$ rozšiřovat, protože jakýkoliv další vzor by definoval stejnou podmnožinu X, jakou již definuje nějaký prvek $\mathcal{L}$. Z těchto důvodů se tedy zdá, že ideální volbou zaujetí je co nejmenší (nejsilnější) $\mathcal{L}$, které zmíněnou podmínku splňuje. Zde je ovšem kámen úrazu, neboť jednoduchými argumenty (Mitchell, 1980) lze ukázat, že takové zaujetí $\mathcal{L}$ je stále příliš slabé. Použitím $\mathcal{L}$ nebude klasifikační algoritmus nikdy schopen rozhodnout o třídě kteréhokoliv objektu $x \in X \setminus D$. Skutečně, při hlasování vzorů z T o třídě tohoto prvku dojde k remíze!

Pro rozumný výsledek je tedy nutné zesílit zaujetí, tedy uvažovat nějaký jazyk $\mathcal{L}$ takový, že $\{\phi(X) \mid \phi \in \mathcal{L}\} \subset 2^X$. Algoritmus tedy z principu nemůže být „nestranný", některá možná rozdělení objektů na kladné a záporné musí předem zavrhnout. Tím je dobře odůvodněna volba termínu *zaujetí* pro $\mathcal{L}$. Které vzory ϕ ale předem zatratit, aniž bychom vynechali vzory dobře odpovídající skutečnosti? Na tuto otázku už nenalezneme odpověď v rámci procesu získávání znalostí. Vhodné vymezení $\mathcal{L}$ je dáno naší vstupní znalostí o problému.

Uvedené úvahy o volbě zaujetí můžeme za jednoduchých podmínek zobecnit z konkrétní úlohy těžení klasifikačních vzorů na obecnou úlohu získávání znalostí, v níž budeme uvažovat *konečný* jazyk $\mathcal{L}$. Konečnost jazyka pro nás nepředstavuje závažnou

újmu na obecnosti, neboť mlčky předpokládáme, že úlohu získávání znalostí vykonává nějaký algoritmus. Ten v konečném čase stejně vyhodnotí jen konečný počet vzorů. V dalších oddílech budeme uvažovat i nekonečné množiny vzorů, například vzory parametrizované nějakou reálnou proměnnou θ. Budeme mít ovšem na paměti, že taková množina je ve skutečnosti pouze *aproximována* nějakým konečným jazykem $\mathcal{L}$ obsahujícím vzory například pro konečný vzorek hodnot parametru θ. Čím je $|\mathcal{L}|$ větší, tím může být aproximace přesnější.

Budeme znovu předpokládat nějakou (pro jednoduchost spočetnou) množinu X, ale nebudeme se už zabývat klasifikací, tj. jejím dělením na kladné a záporné prvky. Předpokládejme pouze, že existuje nějaká funkce $\zeta: \mathcal{L} \times X \to \{0, 1\}$ vyhodnocující to, zda vzor $\phi \in \mathcal{L}$ je v souladu s prvkem $x \in X$ (x můžeme považovat za nějaké konkrétní pozorování). Pokud například ϕ bude vyjadřovat, že lidé za deště nosí deštníky a x bude popis situace, kdy někdo stál v dešti bez deštníku, pak $\zeta(\phi, x) = 0$. Uvažujme ještě, že pozorování x nastávají s určitými pravděpodobnostmi, tedy že je definováno pravděpodobnostní rozdělení P_X. Nyní můžeme definovat kvalitu vzoru ϕ jako střední hodnotu jeho souladu s prvky X:

$$Q(\phi) = \sum_{x \in X} \zeta(\phi, x) P_X(x) . \tag{9.3}$$

Nejkvalitnějším vzorem je tedy

$$\phi^* = \arg\max_{\phi \in \mathcal{L}} Q(\phi) . \tag{9.4}$$

Data $D = \{x_1, x_2, ..., x_m\}$ budou nyní obsahovat náhodně a navzájem nezávisle vylosované prvky z X (D je tedy multimnožina, některé objekty z X může obsahovat vícekrát). Hledejme vzor $\hat{\phi}$ tentokrát podle rovnice (9.2), v níž vyhodnocovací funkce q je přirozeně definována jako

$$q(\phi, D) = \frac{1}{m} \sum_{i=1}^{m} \zeta(\phi, x_i) . \tag{9.5}$$

Samozřejmě bychom rádi, aby vzor $\hat{\phi}$ nalezený dataminingovým algoritmem podle rovnice (9.2) byl právě tím nejlepším vzorem ϕ^* určeným rovnicí (9.4). To se ovšem nepodaří vždy, ale chtěli bychom alespoň, aby $\hat{\phi}$ nebyl o mnoho horší než ϕ^*, tedy aby hodnota $Q(\phi^*) - Q(\hat{\phi})$ nebyla příliš velká. Jak toho dosáhnout? Předpokládejme prozatím bez důvodu, že q dobře odhaduje kvalitu vzorů, tedy že pro nějaké $\varepsilon > 0$ a nějaký vzor $\phi \in \mathcal{L}$ platí

$$q(\phi, D) - Q(\phi) \le \varepsilon , \tag{9.6}$$

$$Q(\phi) - q(\phi, D) \le \varepsilon . \tag{9.7}$$

Vzhledem k rovnici (9.2) platí $q(\phi^*, D) \le q(\hat{\phi}, D)$, a tedy z nerovnosti (9.6) máme

$$q(\phi^*, D) - Q(\hat{\phi}) \le \varepsilon . \tag{9.8}$$

Protože nerovnost (9.7) platí pro všechna $\phi \in \mathcal{L}$ včetně ϕ^*, je $q(\phi^*, D) \geq Q(\phi^*) - \varepsilon$. Využitím tohoto spodního odhadu v nerovnici (9.8) dostáváme kýžené omezení

$$Q(\phi^*) - Q(\hat{\phi}) \leq 2\varepsilon . \tag{9.9}$$

Zbývá ovšem zajistit platnost nerovnic (9.6) a (9.7), které lze ekvivalentně zapsat jako

$$\left| q(\phi, D) - Q(\phi) \right| \leq \varepsilon . \tag{9.10}$$

Pro daný vzor ϕ závisí platnost nerovnice (9.10) na datech D. Data jsou generována náhodně z rozdělení P_X, a tedy platnost nerovnosti je také náhodným jevem. Lze ukázat[1], že pro pevně zvolený vzor ϕ bude nerovnost porušena nejvýše s pravděpodobností $2e^{-2\varepsilon^2 m}$ (připomeňme, že $m = |D|$). Pravděpodobnost, že nerovnost nebude platit pro některý vzor $\phi \in \mathcal{L}$, je tedy nejvýše δ, kde

$$\delta = 2\left| \mathcal{L} \right| e^{-2\varepsilon^2 m} .$$

Pokud z této rovnice vyjádříme ε a dosadíme je do nerovnice (9.9), odvodíme, že s pravděpodobností alespoň $1 - \delta$ se kvalita získaného vzoru ϕ bude od kvality nejlepšího ϕ^* vzoru v $\mathcal{L}$ lišit takto

$$Q(\phi^*) - Q(\phi) \leq 2\sqrt{\frac{1}{2m} \ln \frac{2|\mathcal{L}|}{\delta}} . \tag{9.11}$$

Narazili jsme tedy na zřejmé dilema charakteristické pro volbu $\mathcal{L}$. Se slabým zaujetím algoritmu můžeme dle rovnice (9.4) dosáhnout vysoké kvality nejlepšího vzoru ϕ^* v $\mathcal{L}$, neboť maximalizujeme přes velkou množinu $\mathcal{L}$. Na druhou stranu pak podle nerovnosti (9.11) musíme očekávat velký rozdíl mezi kvalitou tohoto nejlepšího vzoru ϕ^* na jedné straně a kvalitou vzoru $\hat{\phi}$ skutečně nalezeného algoritmem na straně druhé. Pokud o nějaké podmnožině $\mathcal{L}' \subset \mathcal{L}$ víme, že obsahuje ϕ^*, je z tohoto hlediska vhodné zesílit zaujetí z $\mathcal{L}$ na $\mathcal{L}'$. Nejlepší výsledek bychom obdrželi v extrémním případě, kdy $\mathcal{L}' = \{\phi^*\}$, ale v takovém případě je již naše apriorní znalost totožná s výsledkem „těžení".

Ukázali jsme tedy zatím obecně, že apriorní znalosti promítáme do procesu získávání znalostí stanovením zaujetí algoritmu $\mathcal{L}$, přičemž užitečnost této znalosti roste s kvalitou vzorů obsažených v $\mathcal{L}$ a klesá s velikostí této množiny. V dalších oddílech kapitoly probereme některé tradiční techniky pro popis $\mathcal{L}$ z (neformální) znalosti datové domény.

Předtím se ještě krátce dotkněme otázky, zda existuje metoda, která by přijala dvě zaujetí $\mathcal{L}_1$, $\mathcal{L}_2$ a bez jakékoliv další informace (dat D a doménové znalosti) rozhodla, které z nich je lepší, např. v tom smyslu, že povede k lepším výsledkům ve více úlohách získávání znalostí. Podle příspěvku (Shaffer, 1994) taková metoda neexistuje, neboť, volně řečeno, všechna zaujetí jsou si co do kvality rovna přes všechny možné datové

[1] Využijeme přitom tzv. Hoeffdingovy nerovnosti (Hoeffding, 1963) a (Schlesinger, Hlaváč, 1999).

domény a úlohy. Tento závěr samozřejmě nebere v úvahu, že různé datové domény a úlohy se v reálném světě nevyskytují se stejnou pravděpodobností. Mnohé techniky v data miningu používají princip známý jako *Ockhamova břitva*, tedy vycházejí z předpokladu, že jednodušší vzory modelují reálná data obvykle lépe než vzory složitější. Ačkoliv se tento předpoklad nepotvrzuje v experimentech klasifikačního strojového učení (Zahálka, Železný, 2011), jednodušší vzory minimálně nabízejí snazší interpretovatelnost člověkem (Domingos, 1999).

V případě, když se neumíme na základě doménové znalosti rozhodnout mezi dvěma či více zaujetími, ale máme již k dispozici data D, je možné odhadnout, které bude nejvhodnější, a to za pomoci těchto dat. K tomu slouží techniky jako třeba *křížová validace* (Hastie a kol., 2001), kterým se zde ale věnovat nebudeme.

Poznamenejme ještě, že zaujetí $\mathcal{L}$, o němž jsme dosud diskutovali, se někdy označuje jako zaujetí *pevné* (hard bias). Algoritmus může být vybaven i zaujetím *volným* (soft bias) v tom smyslu, že některé vzory bez ohledu na data D *preferuje*. Taková forma zaujetí se snadno formalizuje v případě těžení podle rovnice (9.2), v níž funkci q rozdělíme na součet

$$q(D, \phi) = q_1(D, \phi) + \lambda q_2(\phi), \tag{9.12}$$

kde q_1 měří kvalitu ϕ vzhledem k datům D a q_2 představuje volné zaujetí. Může například nabývat větších hodnot pro syntakticky jednodušší vzory nebo pro vzory, které se jeví jako nadějnější na základě apriorní znalosti. Konstanta $\lambda \in R$ pak váží příspěvek volného zaujetí k celkovému ohodnocení vzoru.

9.2 Parametrické metody

Mnoho typů hledaných ϕ se skládá ze dvou složek *diskrétní struktury* a *reálných parametrů* vztažených k prvkům této struktury. Oblíbený přístup jak stanovit zaujetí spočívá v tom, že struktura se určí předem a z dat ladí algoritmus pouze parametry. Metodám založeným na této strategii se proto říká *parametrické*. Parametrické metody se obvykle používají za účelem hledání jediného výsledného vzoru podle rovnice (9.2).

Co konkrétně může být ona diskrétní struktura? Obecně je to nějaký matematický výraz. Uvažujme například úlohu, v níž prvky dat $D = \{x_1, x_2, ..., x_m\}$ jsou náhodně, navzájem nezávisle vybrány z nějakého pravděpodobnostního rozdělení P_{Z^*} na nezáporných celých číslech a chceme podle rovnice (9.2) najít vzor ve formě funkce $\phi : Z^* \rightarrow [0, 1]$ aproximující toto rozdělení. Vhodnou volbou ohodnocující funkce q je v tomto případě tzv. *věrohodnost*

$$q(D, \phi) = \prod_{i=1}^{m} \phi(x_i). \tag{9.13}$$

Bez apriorní znalosti o P_{Z^*} by řešením rovnice (9.2) byl vzor

$$\hat{\phi}(x) = \frac{N(x)}{m}, \tag{9.14}$$

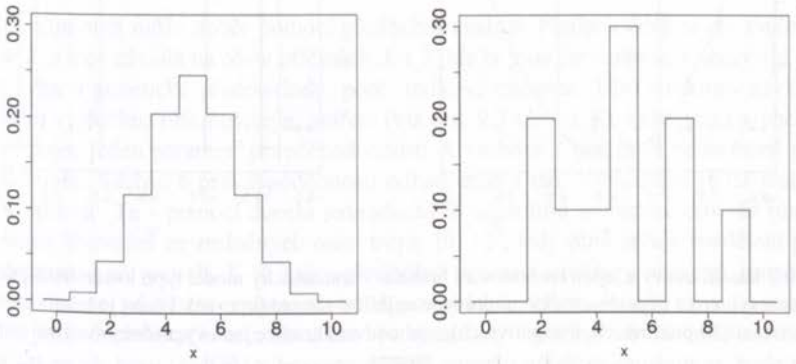
kde $N(x)$ je počet výskytů hodnoty x v D. Toto řešení evidentně pokulhává, neboť přiřazuje nulovou pravděpodobnost všem číslům neobsaženým v D. To je vidět třeba v případě, kdy data jsou ve skutečnosti generována z binomického rozdělení

$$P_{Z'}(x) = \binom{n}{x} p^x (1-p)^{n-x}, \tag{9.15}$$

kde $n = 10$ a $p = 0{,}5$ a konkrétně by třeba nabyla hodnot

$$D = \{5, 4, 7, 2, 7, 2, 5, 5, 9, 3\}. \tag{9.16}$$

Vzor $\hat{\phi}$ nalezený uvedeným způsobem by přiřkl – zjevně proti naší intuici – nulovou pravděpodobnost číslu 6, zatímco pro jeho sousedy 5 a 7 by odhadl pravděpodobnost nenulovou. Pokud ovšem předem víme, že data jsou generována z binomického rozdělení s $n = 10$, určíme[2] strukturu hledaného vzoru podle rovnice (9.15). Zaujetí $\mathcal{L}$ tedy obsahuje reprezentace vzorce (9.15) (s $n = 10$) lišící se hodnotou p, která probíhá nějakou konečnou množinu reálného intervalu [0, 1], například $p = 0{,}1$; 0,2; ...; 1. Rovnici (9.13) potom ve shodě s intuicí maximalizuje vzor[3] s $\hat{p} = 0{,}5$. Situace je ilustrována obrázkem 9.1.



Obr. 9.1 Rozdělení získaná z dat (16) maximalizací věrohodnosti s apriorní znalostí o typu rozdělení (vlevo) a bez ní (vpravo).

Strukturní apriorní znalost se hojně využívá i v klasifikační úloze data miningu již ilustrované v předchozím oddílu. Jednoduchým příkladem takto účinkující struktury je *kvadratická funkce*. Data D budiž nějakou konečnou podmnožinou R a $D_+ \subseteq D$ nechť

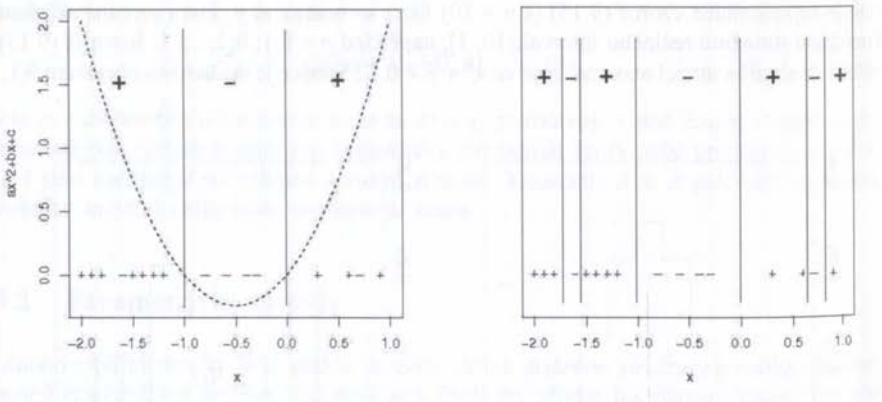
[2] Pokud neznáme ani n, budeme pro strukturu podobně hledat oba neznámé parametry. V konkrétním případě binomického rozdělení pak ale není volba ohodnocovací funkce z rovnice (9.13) nejvhodnější (DasGupta, Rubin, 2005).

[3] Odhad hodnoty parametru p maximalizující věrohodnost se vypočte podle $\hat{p} = \text{round}\left(\frac{1}{mn}\sum_{i=1}^{m} x_i, 1\right) = 0{,}5$, kde funkce round($x$, d) zaokrouhluje reálné x na d desetinných míst.

obsahuje právě všechny prvky D, jejichž třída je kladná. Každý uvažovaný vzor φ nechť definuje nějakou podmnožinu $\phi(R) \subseteq R$. Hledejme ten nejlepší podle vztahu (9.2), kde $q(D, \phi)$ je počet prvků patřících buď do obou, nebo do žádné z množin $\phi(R) \cap D$ či D_+ (maximalizujeme tedy shodu vzoru s klasifikací D_+). Kvadratický prahový vzor ϕ je v tomto případě dán třemi reálnými parametry a, b, c (zaujetí $\mathcal{L}$ lze tedy reprezentovat nějakou konečnou podmnožinou R^3) a je interpretován jako

$$x \in \phi(X) \text{ právě tehdy, když } ax^2 + bx + c > 0.$$

Obrázek 9.2 ilustruje použití kvadratické prahové funkce v konkrétní klasifikační úloze. Demonstruje odlišnosti v rozhodovací hranici mezi parametrickým modelem a jeho neparametrickým protějškem implementovaným metodou nejbližšího souseda. Důsledkem zaujetí kvadratického prahového vzoru je mimo jiné i nekonzistence s množinou D.



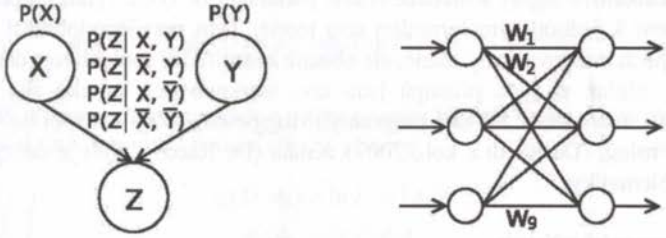
Obr. 9.2 Klasifikátory a jejich rozhodovací hranice. Parametrický model typu kvadratická prahová funkce (vlevo) a neparametrický model typu nejbližší soused (vpravo). Úloha je řešena pro 16 pozorování (10 pozitivních, 6 negativních), rozhodovací hranice jsou vyznačeny svislými plnými čarami.

Pro klasifikační a s ní související *regresní* úlohu (v ní jsou vzory zobrazeními do R) se používá i celá řada jiných struktur, a to jak jednodušších (zejména lineární prahové funkce), tak komplikovanějších (např. mnohočleny vyšších řádů, logistické funkce, splinové funkce). Kniha (Hastie a kol., 2001) dobře vysvětluje mnoho z nich.

9.2.1 Grafové modely

Některé matematické struktury používané v data miningu pro definici zaujetí mají tu výhodu, že je lze vyjádřit *grafem*. Typickým příkladem jsou *bayesovské sítě* (Jiroušek, Vejnarová, 1997), které motivujme příkladem. Sledujeme výskyt tří jevů u pacientů: nemoci Z a jejích dvou možných příčin X, Y. Data $D = \{x_1, x_2, ..., x_m\}$ tedy obsahují binární trojice $x_i \in \{0, 1\}^3$ odpovídající jednotlivým pacientům. Hledaný vzor ϕ má apro-

ximovat skutečné pravděpodobnostní rozdělení těchto trojic. Hledejme jej dle rovnice (9.2) s použitím hodnoticí funkce z rovnice (9.13). Bez předběžné znalosti je opět řešením rovnice (9.14). Vzor je funkcí s definičním oborem o $2^3 = 8$ prvcích. Lze jej tedy uchovat ve formě tabulky o 7 řádcích[4]. Zde je evidentní obecný problém: pokud bychom sledovali n binárních veličin, budeme pro popis vzoru potřebovat řádově 2^n řádků.[5] S rostoucím n tedy tato metoda rychle přestane být paměťově zvládatelná.



Obr. 9.3. Příklady vzorů s grafovou strukturou a reálnými parametry. Vlevo bayesovská síť, vpravo dopředná neuronová síť. Struktura je obvykle navržena na základě předběžné znalosti a parametry jsou hledány pomocí dat.

Zde nám opět může dobře pomoci předběžná znalost. Předpokládejme, že sledovaná nemoc Z je sice závislá na obou příčinách X a Y, ale ty jsou již vzájemně nezávislé. Může jít třeba o genetické předpoklady, popř. rizikové chování. Tuto strukturu závislostí přehledně vyjádříme orientovaným grafem (viz obr. 9.3 vlevo). Ke každému z vrcholů X, Y je připojen jeden parametr pravděpodobnosti, k vrcholu Z pak čtyři podmíněné pravděpodobnosti. Těchto 6 pravděpodobností odhadneme z dat. Výsledkem je již zmíněná bayesovská síť. Ta s pomocí docela jednoduchých algoritmů umožňuje odvodit pravděpodobnost libovolné ze zmíněných osmi trojic $\{0, 1\}^3$, tedy plně určuje rozdělení pravděpodobnosti nad jevy X, Y, Z. Díky předběžné znalosti struktury této sítě jsme z dat odhadovali šest parametrů místo původně potřebných sedmi. Rozdíl není ohromující v tomto jednoduchém příkladě, ale může být významný v případě obecném. Bude-li totiž počet vstupních hran (rodičů) v bayesovské síti omezen nějakou konstantou, bude počet parametrů této sítě růst s n pouze polynomicky. Bayesovské sítě patří do širší třídy tzv. grafických pravděpodobnostních modelů (Bishop, 2006) a (Koller, 2009).

Zaujetí algoritmu ve formě grafové struktury je využíváno i v těžení klasifikačních vzorů. Uveďme alespoň příklad dopředných umělých neuronových sítí, které definují nelineární zobrazení $R^n \to \{0, 1\}^p$ ($p \in N$) vyjádřitelné kombinací grafu a množiny reálných parametrů (viz obr. 9.3 vpravo). Podrobnosti lze najít například v knihách (Hastie a kol., 2001) nebo Umělá inteligence 1 (Hořejš, 1993).

[4] Stačí 7 řádků, neboť $\phi(0, 0, 0) = 1 - \sum_{x \neq [0,0,0]} \phi(x)$.

[5] Stejně rychle porostou požadavky na množství dat potřebných k tomu, aby ϕ dobře aproximovalo skutečné rozdělení trojic.

9.2.2 Logické modely

Grafy představují v data miningu tradiční způsob určení struktury vzoru na základě apriorní znalosti. V nedávno se rozvinutém oboru tzv. *statistického relačního učení* (Getoor, Taskar, 2007) jsou ovšem vyvíjeny další formalismy parametrického učení, jejichž strukturní součásti mají větší výrazovou schopnost než grafy a mohou tedy modelovat složitější apriorní znalost. Zvláštní pozornost zde mají vzory, jejichž struktura je teorií v jazyce predikátové logiky a hledané reálné parametry obvykle vyjadřují pravděpodobnosti vztažené k jednotlivým formulím této teorie. Tyto pravděpodobnosti jsou v různých přístupech interpretovány různě, ale obecně kvantifikují spolehlivost dané formule.

Příklady těchto nových přístupů jsou tzv. markovovské logické sítě (Domingos a kol., 1999), stochastické logické programy (Muggleton, 2003) či ProbLog – pravděpodobnostní Prolog, (De Raedt a kol., 2007). Kniha (De Raedt, 2008) je dobrým úvodem do této problematiky.

9.2.3 Bayesovský přístup

Vedle pevného zaujetí určujícího strukturu vzorů se v parametrických metodách často promítá apriorní znalost i prostřednictvím zaujetí volného, které diktuje preference na určité hodnoty parametrů. Hodnoty všech parametrů vzoru označíme společně symbolem θ a ϕ_θ tak označuje vzor s hodnotami parametrů θ.

Preferenci pro určité θ vyjádříme *apriorní pravděpodobností* $P(\phi_\theta)$ vzoru ϕ_θ. Tato pravděpodobnost je *subjektivní* v tom smyslu, že vyjadřuje naši počáteční důvěru v to, že θ je „správná" parametrizace dané struktury vzoru, a to bez ohledu na data D. Metody využívající apriorních pravděpodobností se obvykle označují jako *bayesovské* (Duda a kol., 2001).

Dále budeme uvažovat věrohodnost $P(D|\phi_\theta)$ vyjadřující pravděpodobnost vygenerování dat D za podmínky, že platí vzor ϕ_θ. Věrohodnost lze definovat rozdílně v různých konkrétních typech dataminingových úloh, s jedním způsobem definice jsme se setkali již v úvodu oddílu 9.2. V klasifikačních úlohách by vhodnou volbou byla definice

$$P(D \mid \phi_\theta) = \begin{cases} 1/|\mathcal{L}| \text{ pokud } \phi_\theta \in \mathcal{L}_k \\ 0 \text{ v ostatních případech} , \end{cases}$$

kde $\mathcal{L}_k \in \mathcal{L}$ je množina všech vzorů z jazyka $\mathcal{L}$ konzistentních s daty D (vlastnost konzistence jsme již používali v příkladech hledání klasifikačních vzorů).

V případě, kdy vzor přímo definuje pravděpodobnosti prvků množiny D, by přirozenou volbou byla definice

$$P(D \mid \phi_\theta) = \frac{q(D, \phi_\theta)}{\sum_{\phi_\theta \in \mathcal{L}} q(D, \phi_\theta)} ,$$

kde funkce q je definována vztahem (9.13) pro $\phi \equiv \phi_\theta$.

Pomocí apriorní pravděpodobnosti $P(\phi_\theta)$ a věrohodnosti $P(D|\phi_\theta)$ můžeme nyní vyjádřit aposteriorní pravděpodobnost vzoru ϕ_θ, tedy jeho pravděpodobnost s ohledem na data D

$$P(\phi_\theta \mid D) = \frac{P(D, \phi_\theta)P(\phi_\theta)}{\sum_{\phi_\theta \in \mathcal{L}} P(D, \phi_\theta)P(\phi_\theta)}. \tag{9.17}$$

Naším cílem je najít vzor maximalizující aposteriorní pravděpodobnost. Přitom platí

$$\arg\max_{\phi_\theta}(P(\phi_\theta \mid D) = \arg\max_{\phi_\theta}(\ln(P(\phi_\theta \mid D) + \ln P(\phi_\theta)) \tag{9.18}$$

a bayesovský přístup je tedy ve shodě s naším pojetím hledání vzoru podle rovnice (9.2), v níž hodnoticí funkce je dána rovnicí (9.12), přičemž

$$q_1(D, \phi_\theta) = \ln P(D \mid \phi_\theta),$$
$$q_2(\phi_\theta) = \ln P(\phi_\theta),$$

kde $q_2(\phi_\theta)$ definuje volné zaujetí algoritmu a $\lambda = 1$. Obecně však můžeme volit $\lambda > 1$, a tím zesílit vliv volného zaujetí.

V aplikacích často nelze z doménově specifické znalosti stanovit konkrétní apriorní rozdělení $P(\phi_\theta)$. V takových případech je obvyklá volba volného zaujetí motivována již zmíněným Ockhamovým principem, tentokrát interpretovaným tak, že pravděpodobnější jsou vzory s menšími absolutními hodnotami parametrů. Často se tedy například pokládá

$$q_2(\phi_\theta) = \ln P(\phi_\theta) = -\|\theta\|,$$

kde $\|\theta\|$ je nějaká norma, např. pro $\theta = [w_1, w_2, \ldots, w_k]$ je $\|\theta\| = \sum_{i=1}^{k} w_i^2$. Pro parametry w_i, které se dostatečně přiblížily k nule, můžeme dále stanovit $w_i = 0$. Následně tak můžeme zjednodušit i strukturu vzoru (například v grafu umělé neuronové sítě odebereme hrany ohodnocené nulovými parametry), a tak volným zaujetím působit na zaujetí pevné. Ilustrované technice se říká *regularizace* a její použití vede ke snížení variance vzorů $\hat{\phi}$ získávaných z různých dat D.

9.3 Neparametrické metody

V neparametrických metodách hledá v rámci zaujetí $\mathcal{L}$ dataminingový algoritmus jak vhodnou diskrétní strukturu vzorů, tak jejich případné reálné parametry.[6] Neparametrické metody jsou v data miningu velmi oblíbené a používají se v obou scénářích vyjádřených rovnicemi (9.1) a (9.2). V prvém případě je ohodnocující predikát q obvykle volen tak, že platí pro vzory ϕ, které se datech D v nějakém smyslu často vyskytují. V druhém případě obvykle q zachycuje, jak kvalitně vzor ϕ data D *modeluje*, např. jde o přesnost klasifikačního vzoru. Strukturami vzorů bývají množiny (např. častých položek), po-

[6] Vžité označení *neparametrické metody* je tedy poněkud zavádějící.

sloupnosti, grafy či logické konstrukty jako výrokové nebo predikátové klauzule. Některé typy struktur mají grafickou i logickou součást (např. tzv. *rozhodovací stromy*).

Apriorní znalost se do volby zaujetí $\mathcal{L}$ promítá zhruba dvěma způsoby. Zaprvé může být použita k formalizaci omezení na syntax vzorů. Zadruhé může být apriorní znalost vyjádřena v samotném jazyce vzorů (nebo jazyce jemu podobném), přičemž hledané vzory mají tuto znalost vhodně doplnit. Kromě toho může být apriorní znalost podobně jako v případě parametrických metod promítnuta i do volby volného zaujetí.

9.3.1 Syntax vzorů

Jazyk $\mathcal{L}$ obsahuje vzory vyhovující nějaké *gramatice*. Tato gramatika ve většině dataminingových algoritmů není vyjádřena explicitně (např. ve formě přepisovacích pravidel), ale je implicitní součástí samotného algoritmu. Uživatel přitom může ovlivnit jen některé její parametry účinkující jako omezující podmínky na $\mathcal{L}$. Touto podmínkou může být třeba maximální velikost popisu vzoru (např. maximální počet podmínek v konjunktivním pravidle). Výjimkami jsou některé algoritmy vyhledávající vzory se složitou syntaxí, které pro definici jazyka $\mathcal{L}$ explicitních gramatik používají autoři prací (Cohen, 1994), (Dehaspe, De Raedt, 1996). V praxi je tedy obvykle volba jazyka $\mathcal{L}$ dána výběrem dataminingového algoritmu a určením uvedených omezujících parametrů.

Neexistuje obecná metoda, podle níž by bylo možné určit vlastnosti jazyka $\mathcal{L}$ na základě neformální doménové znalosti. Algoritmické postupy jsou známy jen pro určení specifických vlastností $\mathcal{L}$ za specifických podmínek. Příkladem je situace, kdy data D mají příznakovou formu. To znamená, že každé $x \in D$:

$$x \in \mathrm{Dom}_1 \times \mathrm{Dom}_2 \times \ldots \times \mathrm{Dom}_n,$$

kde každé Dom_i je nějaká množina představující definiční obor *příznaku i*. Pokud víme, že nějaká podmnožina příznaků $I \subseteq \{1, 2, \ldots, n\}$ je irelevantní vzhledem ke znalosti, kterou chceme z dat vytěžit, přizpůsobíme zaujetí $\mathcal{L}$ tak, aby neobsahovalo vzory, které se na tyto příznaky odkazují. Tento způsob využití apriorní znalosti je triviální, ale málokdy víme o příznaku jistě, že je irelevantní. Izolací nepotřebných příznaků za obecnějších podmínek se zabývají metody *výběru příznaků* (feature selection) (Liu, Motoda, 1994). Za určitých podmínek je výběr příznaků možný i na základě obecnějších apriorních znalostí o statistickém rozdělení dat D (Pudil, Novovičová, 1998). Obecně ovšem využití doménové znalosti pro návrh co nejsilnějšího zaujetí, které stále zahrnuje potřebné vzory, plně závisí na inteligenci vytěžovatele dat.

V neparametrickém data miningu se setkáváme s algoritmy pro *revizi teorie* (theory revision) (Adé a kol., 1994). Takový algoritmus pracuje obvykle podle scénáře daného rovnicí (9.2). Kromě dat algoritmus dostane počáteční vzor (teorii) ϕ_{init} a výsledného vzoru ϕ dosáhne postupnými opravami ϕ_{init}, které vedou ke zlepšení souladu s daty D. Tedy snadným způsobem, jak uplatnit nedokonalé apriorní znalosti, je zakódovat tyto znalosti do počátečního vzoru ϕ_{init}. Revizi teorie můžeme v souladu s dříve zavedenými pojmy chápat jako proces, při němž apriorní znalost definuje zaujetí algoritmu. Skutečně, algoritmus vyhledává vzor v nějakém „syntaktickém okolí" ϕ_{init}, tedy v jazyce

$$\mathcal{L} = \{\, \phi \mid \rho(\phi, \phi_{\mathrm{init}}) < \varepsilon \},$$

kde ρ je nějaká syntaktická metrika na vzorech a $\varepsilon > 0$.

9.3.2 Explicitní vyjádření apriorní znalosti

Zajímavý případ nastává, pokud máme k dispozici syntakticky vymezený jazyk vzorů $\mathcal{L}_\phi$, ale k vytěžování použijeme jazyk

$$\mathcal{L} = \{(\phi, B) | \ \phi \in \mathcal{L}_\phi\},$$

kde B je prvek nějakého jazyka $\mathcal{L}_B$ sloužícího pro explicitní formulaci apriorní doménové znalosti. Může, ale nemusí být $\mathcal{L}_B = \mathcal{L}_\phi$. Pro těžení pak musíme mít samozřejmě k dispozici predikát q ohodnocující formule z $\phi \in \mathcal{L}_\phi \times \mathcal{L}_B$.

Tato situace je zajímavá, pokud jazyky $\mathcal{L}_\phi$, $\mathcal{L}_B$ mají dostatečné vyjadřovací schopnosti, aby mohlo dojít k „interakci" mezi ϕ a B. To je případ *induktivního logického programování* (De Raedt, 2008), v němž ϕ, B, D jsou klauzální teorie (tj. množiny klauzulí) predikátové logiky. Přitom q(D, (ϕ, B)) platí právě tehdy, když $\phi \cup B \vdash D$, kde $\vdash$ je relace dokazatenosti. Požadujeme tedy, aby všechny klauzule v D byly dokazatelné ze sjednocení klauzálních množin ϕ a B.

Pokud $D \in \mathcal{L}_\phi$ je triviálním řešením $\phi = D$. To se obvykle nepřipouští, což zajistíme tak, aby $D \notin \mathcal{L}_\phi$, třeba požadavkem $|\phi| < |D|$ pro všechna $\phi \in \mathcal{L}_\phi$.

Ilustrujme induktivní logické programování na příkladu. O doméně lidské smrtelnosti předem víme B, kde

$$B = \{\forall x : \text{člověk}(x) \rightarrow \text{smrtelný}(x)\}.$$

Naše data D dále uvádějí, že Sokratés a Aristotelés byli muži a v důsledku toho byli smrtelní

$$D = \{\text{muž(Sokratés)} \rightarrow \text{smrtelný(Sokratés)},\ \text{muž(Aristotelés)} \rightarrow \text{smrtelný(Aristotelés)}\}.$$

Jedním z vyhovujících vzorů je potom

$$\phi = \{\forall x : \text{muž}(x) \rightarrow \text{člověk}(x)\}.$$

Skutečně, $\phi \cup B \vdash D$.

Uvedený způsob určení pevného zaujetí algoritmu pomocí apriorní znalosti má oproti ostatním zde diskutovaným způsobům zásadní výhodu v tom, že poskytuje uživateli komfort dedikovaného formálního jazyka $\mathcal{L}_B$ pro explicitní vyjádření apriorní znalosti B. Metody spočívající ve vymezení syntaxe jazyka $\mathcal{L}$ na základě neformální apriorní znalosti takto přímočarou možnost evidentně neposkytují.

Nabízí se otázka, zda lze jako $\mathcal{L}_B$ použít místo predikátové logiky jiný jazyk, více používaný ve znalostním inženýrství. Konkrétními příklady jsou *ontologické jazyky* (zejména deskripční logiky) v současnosti velmi často používané díky rozmachu sémantického webu. Jejich hlavní výhodou oproti predikátové logice je rozhodnutelnost a často i dostupnost rychlých inferenčních mechanismů. Pokud za $\mathcal{L}_\phi$ ponecháme nějakou podmnožinu jazyka predikátové logiky (např. Hornovy kauzální teorie), pak jeho kombinací s ontologickým jazykem $\mathcal{L}_B$ vznikne hybridní jazyk $\mathcal{L}_\phi \times \mathcal{L}_B$. Hlavním problémem pro těžení vzorů je potom vymyslet vhodný dokazovací mechanismus (potřebný pro vyhod-

nocovací predikát q) pro tento hybridní jazyk. Výzkum v tomto směru představuje například článek (Lisi, 2008). Další přirozenou možností je použít ontologického jazyka jak pro $\mathcal{L}_B$, tak pro $\mathcal{L}_\phi$, čímž se vyhneme hybridním jazykům. Tato strategie je zkoumána například v práci (Lehmann, Hitzler, 2010).

9.3.3 Volné zaujetí

V odstavci 9.2.3 jsme ukázali bayesovský přístup ke stanovení volného zaujetí dataminingového algoritmu v kontextu parametrických metod. Analogický přístup lze použít i v rámci metod neparametrických. Zde místo apriorního rozdělení pravděpodobnosti na hodnotách parametrů uvažujeme apriorní rozdělení pravděpodobnosti úplně popsaných vzorů.

Příkladem tohoto přístupu v neparametrickém kontextu, konkrétně v induktivním logickém programování, je práce (Mitchell, 1980). Zde je stanoveno apriorní rozdělení $P(\phi)$ na vzorech ϕ, jimiž jsou klauzální logické teorie. Toto rozdělení nevychází ze specifické doménové znalosti, ale je opět inspirováno principem Ockhamovy břitvy. Konkrétně $P(\phi)$ klesá exponenciálně se syntaktickou složitostí teorie ϕ.

Jak už jsme ukázali v odst. 9.2.3, uplatnění Ockhamovy břitvy při definici apriorního rozdělení $P(\phi)$ vede na princip regularizace, tedy hodnocení vzorů podle rovnice (9.12), v níž funkce q_2 nabývá větších hodnot pro jednodušší vzory. Mnoho algoritmů neparametrického data miningu skutečně takového regularizačního principu využívá, ať už jde o zmíněné induktivní logické programování, nebo třeba o těžení rozhodovacích stromů, jež jsou cíleně zjednodušovány („prořezávány") i za cenu snížení jejich souladu s daty.

Kromě regularizace lze ovšem funkci q_2 stanovit přímo z doménově specifické znalosti. Například v práci (Kléma a kol., 2008) jsou hledány množiny často společně exprimovaných genů. Přitom ohodnocení každé množiny není jen funkcí dat D, ale stoupá i s funkční podobností genů zjištěnou z apriorní znalostní báze (genové ontologie).

Poměrně novým přístupem k využití doménové znalosti je tzv. *učení s přenosem* (transfer learning), v němž se znalost získaná těžením v jedné datové doméně přenáší do domén jiných. Přístup vychází z předpokladu, že pokud byl v jedné doméně nalezen dobrý vzor $\hat{\phi}$, tak v jiných, ale podobných doménách budou nadějné vzory, které jsou $\hat{\phi}$ v nějakém smyslu podobné. Tato obecná myšlenka byla využita v mnoha konkrétních variantách učení s přenosem, viz např. přehledovou studii (Pan a Yang, 2010). V jednoduchém případě si ale učení s přenosem můžeme představit jako způsob stanovení volného zaujetí. Například můžeme stanovit $q_2(\phi) = -\rho(\phi, \hat{\phi})$, kde ρ je nějaká metrika měřící odlišnost vzorů a $\hat{\phi}$ je vzor nalezený v dataminingovém problému podobném tomu aktuálnímu.

9.4 Metaznalosti

V předchozích oddílech jsme předpokládali, že apriorní znalost se týká konceptů samotné datové domény. Promítali jsme ji do zaujetí dataminingového algoritmu, tedy volně řečeno, radili jsme algoritmu, jak mají vypadat těžené vzory. Naše apriorní znalost se

ovšem může týkat i předpokládaných vlastností samotného algoritmu v dané datové doméně. Na základě této *metaznalosti* můžeme, opět volně řečeno, dataminingovému algoritmu poradit, jak vzory hledat.

Příkladem této myšlenky je práce (Žáková, Železný, 2007), která rozpracovává jednoduché pozorování, jež lze ilustrovat tímto způsobem. Předpokládejme, že algoritmus vyhodnotí vzor $\phi \in \mathcal{L}$:

$$\phi = \mathsf{a} \rightarrow \mathsf{b}$$

(tj. a implikuje b) jako nespolehlivý v datech D v tom smyslu, že D popisují příliš mnoho případů, kde platí a a neplatí b. Pokud zároveň z naší doménové znalosti vyplývá, že

$$\mathsf{c} \rightarrow \mathsf{b}, \tag{9.19}$$

tak je předem zřejmé, že i vzor $\phi = \mathsf{a} \rightarrow \mathsf{c}$ bude nespolehlivý, aniž by jej algoritmus musel vyhodnocovat vzhledem k D. V práci (Žáková, Železný, 2007) jsou apriorní znalosti jako (9.19) čerpány z doménové ontologie a využity již ilustrovaným způsobem ke zrychlení algoritmu.

Naznačenou techniku můžeme zobecnit v tom smyslu, že rozborem podmínek na vzory, jež implementuje predikát q, chceme identifikovat podmnožiny $\mathcal{L}(\phi') \subseteq \mathcal{L}$ vzorů ϕ, které určitě nesplní q, pokud jej nesplnil vzor ϕ'. Jakmile v průběhu hledání vzorů algoritmus zjistí, že nějaký vzor ϕ' nesplňuje q, nemusí se již zabývat vzory v $\mathcal{L}(\phi')$. Tímto problémem se zabývá obor označovaný jako *data mining s omezujími podmínkami* (constraint-based data mining) (Bouicaut, Jeudy, 2005).

Další využití metaznalosti se nabízí v kontextu poněkud širšího pojetí data miningu, než jak jsme jej definovali rovnicemi (9.1) a (9.2). V praktických experimentech je obvykle nutné mimo samotné vyhledávání vzorů popsané těmito rovnicemi uskutečnit další procedury týkající se například změny reprezentace vstupních dat, výběru jejich instancí, konstrukce příznaků atd. Sestavit vhodnou posloupnost těchto procedur a přiřadit jim vhodné implementace algoritmů může představovat celkem složitý optimalizační problém. Ten je možné řešit pomocí algoritmů pro automatické plánování. Jedna z prací týkajících se tohoto problému je (Žáková, Železný, 2010). Zde je plán dataminingového procesu sestaven na základě apriorní znalosti o algoritmech a datových a znalostních strukturách. Tato apriorní znalost je zachycena ve formě ontologie.

Jeden ze způsobů jak získat potřebnou metaznalost spočívá v rozboru dřívějších dataminingových experimentů. Data, která dataminingové experimenty popisují, mohou sama být analyzována s pomocí dataminingového algoritmu. Jím vyhledané vzory pak vyjadřují znalosti o chování různých dataminingových algoritmů na různých typech dat. Tato strategie je známa jako *metaučení* (Brazdil a kol., 2009). Takto získané metaznalosti mohou být zařazeny do dataminingové ontologie a následně využity pro již zmíněné plánování dataminingových procesů. Tento cíl sleduje například práce (Hilario a kol., 2009).

Ontologie zachycující metaznalosti může být rovněž kombinována s ontologií nesoucí doménovou apriorní znalost. Tuto kombinaci využívá například studie (Kliegr a kol., 2009) jako součást systému pro inteligentní prezentaci dataminingových výsledků.

9.5 Závěrečná poznámka

Tato kapitola přináší jednotící pohled na dolování dat jako na proces využívání znalostí pro získání znalostí nových. Z důvodu různorodosti znalostních formalismů pracujeme s obecnými pojmy, jimiž jsou jazyk, vzor, teorie či zaujetí. Doufáme, že pro čtenáře obeznámeného se základy tradičních algoritmů strojového učení a data miningu přináší tento text jiný úhel pohledu na problematiku získávání znalostí z dat, který je odlišný od obvyklého dělení na metody s učitelem, bez učitele či metody predikční nebo deskripční.

Domníváme se, že základní rozdělení metod strojového učení na parametrické a neparametrické dobře zpřehledňuje výklad technik využívání apriorní znalosti. Samozřejmě se tímto rozdělením dopouštíme určitých nepřesností. Například bayesovské sítě jsme zařadili mezi parametrické metody, neboť jsme předpokládali, že struktura sítě je předem dána (odráží apriorní znalost) a uvažujeme výhradně úlohu učení parametrů. Existují samozřejmě i algoritmy pro učení struktury sítě. Jejich použití pak vede na úlohu neparametrickou, kdy na základě volného zaujetí dle rovnice (9.12) a regularizace upřednostňujeme jednodušší struktury s dostatečnou věrohodností.

Literatura

Adé H., Malfait B., De Raedt L.: RUTH: an ILP theory revision system. In: *Methodologies for Intelligent Systems.* Springer, 1994.

Berka P.: *Dobývání znalostí z databází.* Academia, 2003.

Bishop C. M.: Graphical Models, *Pattern Recognition and Machine Learning*, Springer, 2006.

Boulicaut J.-F., Jeudy B.: *The Data Mining and Knowledge Discovery Handbook*, Constraint-Based Data Mining. Springer, 2005.

Brazdil P., Giraud-Carrier C., Soares C., Vilalta R.: *Metalearning*. Springer, 2009.

Cohen W. W.: Grammatically biased learning: Learning logic programs using an explicit antecedent description language, *Artificial Intelligence*, 68(2), 1994, s. 303–366.

DasGupta A. Rubin H.: Estimation of binomial parameters when both n, p are unknown, *Journal of Statistical Planning and Inference*, 130(1–2): 2005, s. 391–404.

De Raedt L., Kimmig A., Toivonen H.: A probabilistic Prolog and its application in link discovery. In: *Proceedings of the 20th International Joint Conference on Artificial Intelligence* (IJCAI'07), 2007.

De Raedt L.: *Logical and Relational Learning*. Springer, 2008.

Dehaspe L., De Raedt L.: Dlab: A declarative language bias formalism. In: *Proceedings of the International Symposium on Methodologies for Intelligent Systems* (ISMIS'96). Springer, 1996.

Dehaspe L., De Raedt L.: Clausal discovery, *Machine Learning*, 26(2/3), 1997, s. 99–146.

Domingos P., Kok S., Poon H., Richardson M., Singla P.: Unifying logical and statistical AI. In: *Proceedings of the 21st national conference on Artificial intelligence*. AAAI Press, 2006.

Domingos P.: The role of Occam's razor in knowledge discovery, *Data Mining and Knowledge Discovery*, 3, 1999, s. 409–425.

Duda R. O., Hart P. E., Stork D. G.: *Pattern Classification*. Wiley, 2001.

Getoor L., Taskar B., eds: *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

Hastie T., Tibshirani R., Friedman J.: *The Elements of Statistical Learning*. Springer, 2001.

Hilario M., Kalousis A., Nguyen P., Woznica A.: A data mining ontology for algorithm selection and meta-mining. In: *SoKD'09, ECML/PKDD'09 Workshop on Third Generation Data Mining*: Towards Service-oriented Knowledge Discovery, 2009.

Hoeffding W.: Probability inequalities for sums of bounded random variables, *Journal of the American Statistical Association*, 58(301):13–30, 1963.

Hořejš J.: Neuronové sítě. In: *Umělá inteligence 1*. Academia, Praha 1993.

Jiroušek R., Vejnarová J.: Bayesovské. sítě. In: *Umělá inteligence 2*. Academia, Praha 1997.

Kléma J., Blachon S., Soulet A., Cremilleux B., Gandrilon O.: Constraint-based knowledge discovery from sage data, *Silico Biology*, 8(14), 2008, s. 391–404.

Kliegr T., Ralbovský M., Svátek V., Šimůnek M., Jirkovský V., Nemrava J., Zemánek J.: Semantic analytical reports: A framework for post-processing data mining results. In: *Foundations of Intelligent Systems*. Springer, 2009.

Koller F.: *Probabilistic Graphical Models*. MIT Press, 2009.

Lehmann J., Hitzler P.: Concept learning in description logics using refinement operators, *Machine Learning*, 78, 2010, s. 203–250.

Lisi F. A.: Building rules on top of ontologies for the semantic web with inductive logic programming, *Theory and Practice of Logic Programming*, 8(3), 2008, s. 271–300.

Liu H., Motoda H., eds: *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Springer, 1994.

Mannila H.: Aspects of data mining. In: *Proceedings of the MLnet Familiarization Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*, 1995, s. 1–6.

Mitchell T. M.: *The need for biases in learning generalizations*. Technical Report CBM-TR 5-110, Rutgers University, New Brunswick, NJ, 1980.

Muggleton S.: Learning from positive data. In: *ILP '96 Selected Papers from the 6th International Workshop on Inductive Logic Programming*. Springer, 1997.

Muggleton S.: Learning structure and parameters of stochastic logic programs. In: *Proceedings of the 12th international conference on inductive logic programming* (ILP'02). Springer, 2003.

Pan S. J., Yang Q.: A survey on transfer learning, *IEEE Transactions on Knowledge and Data Engineering*, 22(10), 2010, s. 1345–1359.

Pudil P., Novovičová J.: Novel methods for subset selection with respect to problem knowledge, *IEEE Intelligent Systems*, 13(2), 1998.

Shaffer C.: A conservation law for generalization performance. In: *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.

Schlesinger M. I., Hlaváč V.: *Deset přednášek z teorie statistického a strukturního rozpoznání. Učení v rozpoznávání*. Nakladatelství ČVUT, 1999.

Zahálka J., Železný F.: An experimental test of Occam's razor in classification, *Machine Learning*, 82(3), 2011.

Žáková M., Železný F.: Exploiting term, predicate, and feature taxonomies in propositionalization and propositional rule learning. In: *ECML/PKDD'07: 18th European Conference on Machine Learning/11th European Conference on Principles and Practice of Knowledge Discovery*. Springer, 2007.

Žáková M., Křemen P., Železný F., Lavrač N.: Automating knowledge discovery workflow composition through ontology-based planning, *IEEE Transactions on Automation Science and Engineering*, 8(2), 2011.

10 Adaptivní přístupy k dobývání znalostí

10.1 Úvod

Pojmy *adaptivní systém* a *učící se systém* lze nalézt v teorii řízení již v šedesátých letech dvacátého století. V uvedeném kontextu jde v obou případech o systémy, které mají schopnost přizpůsobovat se prostředí, v němž pracují. Tato schopnost je chápána jako minimalizace ztráty, které se systém dopouští v procesu rozhodování (klasifikace). Zatímco adaptivní systém se každé nové situaci přizpůsobuje nezávisle na předcházejících adaptacích a situacích, cílem učícího se systému je dosáhnout dlouhodoběji optimálního chování; zkušenost z jednotlivých dílčích adaptací se zobecňuje do obecněji použitelného modelu. Formálně lze tedy adaptivní systém nebo učící se systém popsat jako (Kotek a kol., 1980) systém se dvěma vstupy:

- projevy prostředí $\boldsymbol{x} \in X$,
- informací o požadovaném chování systému $\Omega \in O_1$,

a jedním výstupem

- reakcí systému $\omega \in O_2$ na dvojici $[\boldsymbol{x}, \Omega]$.

Reakce je dána funkcí f takovou, že $\omega = f(\boldsymbol{x}, q)$, kde q je parametr systému. Každá reakce systému může způsobit ztrátu $Q(\boldsymbol{x}, \Omega, q)$. V průběhu *adaptace* systém pro každou dvojici

$$[\boldsymbol{x}, \Omega]$$

hledá takový parametr q^*, pro který

$$Q(\boldsymbol{x}, \Omega, q^*) = \min_q Q(\boldsymbol{x}, \Omega, q),$$

v průběhu *učení* systém na základě trénovací množiny

$$[\boldsymbol{x}_k, \Omega_k]$$

hledá takový parametr q^*, pro který

$$J(q^*) = \min_q J(q),$$

kde $J(q)$ je střední ztráta

$$J(q) = \int Q(\boldsymbol{x}, \Omega, q)\, \mathrm{d}F(\boldsymbol{x}, \Omega).$$

Učící se systém tedy v teorii řízení představuje dokonalejší nástroj než systém adaptivní.

V poněkud jiném významu se pojem „adaptivní" objevil v posledních letech v souvislosti s inteligentními systémy, zejména se systémy z oblasti strojového učení a soft computingu, kde schopnost učení je většinou nedílnou součástí uvedených metod. V kontextu inteligentních systémů je tedy adaptivita „něco navíc". Evropský výzkumný projekt EUNITE (IST-2000-29207), řešený v letech 2001–2003, chápe adaptivitu jako (Anguita, 2001):
– schopnost inteligentního systému přizpůsobit se změnám prostředí,
– schopnost inteligentního systému přizpůsobit se novým podmínkám využívání,
– schopnost inteligentního systému přizpůsobit se nové aplikaci.

V prvním případě musí být inteligentní systém schopen rozpoznat časové nebo prostorové změny v prostředí, ve kterém působí, a adekvátně reagovat. Příkladem může být rozpoznání změny preferencí zákazníků při elektronickém obchodování nebo regulace nestacionárního systému. Ve druhém případě jde o schopnost systému pracovat i v jiných podmínkách, než pro které byl vyvinut. Příkladem může být možnost přenést systém pro klasifikaci rizikovosti klientů z jedné finanční instituce (např. banky) do jiné finanční instituce (např. pojišťovny) bez nutnosti vývoje celého systému znovu. Třetí případ se týká možnosti přenesení inteligentního systému do zcela jiné aplikační oblasti.

V této kapitole se zaměříme na adaptivní rysy systémů z oblasti strojového učení. Strojové učení přitom budeme chápat jako empirické metody učení se konceptům na základě příkladů rozhodnutí, popř. na základě pozorování. Ve všech sledovaných metodách budeme předpokládat existenci trénovací množiny, která je tvořena konečným počtem příkladů zařazených do některého z konceptů. Zaměříme se tedy na metody učení s učitelem. Na obecné úrovni můžeme toto učení chápat jako prohledávání prostoru možných modelů (obecných popisů jednotlivých konceptů) s cílem nalézt model optimalizující předem zadané kritérium kvality. Tento obecný princip má u jednotlivých metod nejrůznější podobu. U vybraných metod nejprve uvedeme základní podobu „klasického" dávkového učení, kdy má učící se algoritmus k dispozici celou trénovací množinu. Pak se podrobněji podíváme na problémy související s:
– inkrementálním učením,
– učením v měnícím se prostředí,
– učením a zapomínáním,
– integrací znalostí,
– metaučením,
– revizí znalostí,
– využitím analogií.

Uvidíme, že některé standardně používané metody strojového učení přímo vyhovují některému rysu adaptivního chování, jiné metody je třeba vhodně modifikovat.

10.2 Strojové učení a dobývání znalostí

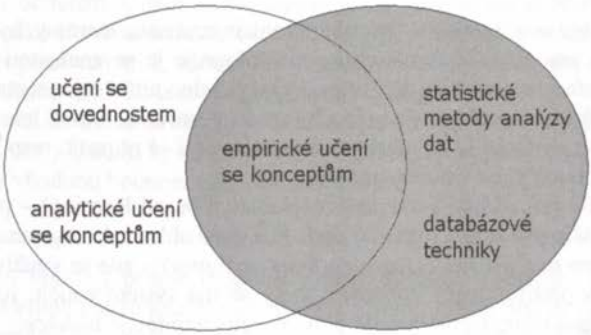
Důležitou vlastností živých organismů je schopnost přizpůsobovat se měnícím se podmínkám (adaptovat se), eventuálně se učit na základě vlastních zkušeností. Schopnost učit se bývá někdy dokonce považována za definici inteligence. Je proto přirozené, že vybavit touto vlastností i systémy technické je jedním z cílů umělé inteligence.

V literatuře lze nalézt řadu definic strojového učení:

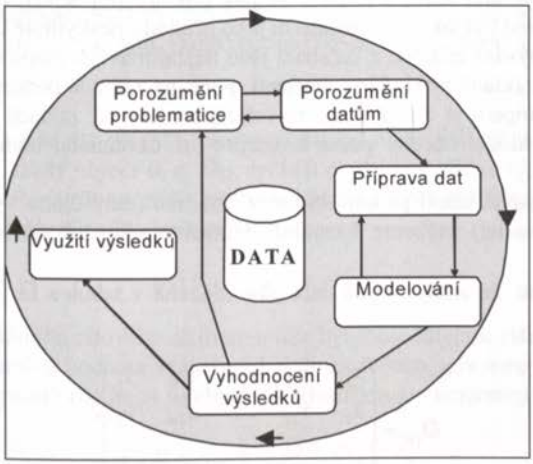
- „The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience" (Mitchell, 1997).
- „Things learn when they change their behavior in a way that makes them perform better in a future" (Witten a Frank, 1999).

V tomto (širším) pojetí učení nenalezneme žádnou zmínku o datech – učící se systém se prostě nějakým způsobem zdokonaluje. Jinak je tomu v definicích dobývání znalostí z databází, kde data jsou východiskem celého procesu:

- „Non-trivial process of identifying valid, novel, potentially useful and ultimately understandable patterns from data" (Fayyad a kol., 1996).
- „Analysis of observational data sets to find unsuspected relationships and summarize data in novel ways that are both understandable and useful to the data owner" (Hand a kol., 2001).



Obr. 10.1 Vztah mezi metodami strojového učení a metodami dobývání znalostí.



Obr. 10.2 Metodika CRSIP-DM.

Zdálo by se tedy, že dobývání znalostí je jen podoborem strojového učení. Není tomu ale tak; jednak se pro dobývání znalostí používají i algoritmy z oblasti statistiky, popřípadě databází (vztah mezi strojovým učením a algoritmy pro dobývání znalostí schematicky ukazuje obr. 10.1), jednak se dobývání znalostí nesoustředí pouze na aplikaci algoritmů na data, ale zaměřuje se i na kroky, které předcházejí nebo následují (obr. 10.2 ukazuje schéma procesu dobývání znalostí z databází dle metodiky CRISP-DM).

V zásadě lze rozlišit dva typy učení: učení se znalostem (knowledge acquisition) a učení se dovednostem (skill refinement). První typ hledá koncepty, obecné zákonitosti apod. (např. jak rozpoznat defraudanta), u druhého typu jde o to zdokonalit své schopnosti na základě procvičování nějaké činnosti (např. jak nalézt cestu v bludišti).

Podíváme-li se na různé metody učení z hlediska úsilí, které je třeba vynaložit na získání nových znalostí, popř. dovedností, lze rozlišovat mezi (Michalski a kol., 1983):

1. *Učením zapamatováním* (rote learning) neboli biflování – systém pouze zaznamenává data nebo dílčí znalosti dodané externím zdrojem; neprovádí se žádná transformace.

2. *Učením se z instrukcí* (learning from instruction, learning by being told) – systém získává znalosti z externího zdroje a integruje je se znalostmi již získanými; provádí se transformace znalostí ze vstupního jazyka do vnitřní reprezentace.

3. *Učením se z analogie* (learning by analogy, instance-based learning, lazy learning) – získávání znalostí je založeno na zapamatování si případů, resp. situací podobných těm, které bude třeba v budoucnu řešit.

4. *Učením na základě vysvětlení* (explanation-based learning) – při učení se využívá několik málo příkladů a rozsáhlé znalosti z dané oblasti (background knowledge).

5. *Učením se z příkladů* (*learning from examples*) – zde se využívá velké množství příkladů (a protipříkladů) konceptu, který se má systém naučit, role background knowledge naopak ustupuje do pozadí; používanou metodou je *indukce*.

6. *Učením se z pozorování a objevování* (learning from observation and discovery) – opět se pracuje s velkým množstvím dat (tedy za využití indukce); systém si ale často sám musí vytvářet koncepty, které se pak pokouší popisovat, navíc data získaná pozorováním nemusí být tak reprezentativní jako příklady poskytnuté učitelem.

Z pohledu dobývání znalostí z databází jsou nejzajímavější empirické metody učení se konceptům na základě příkladů rozhodnutí, popř. na základě pozorování a objevování. Použitým přístupem je zde induktivní inference, kdy na základě konečného počtu příkladů budeme hledat obecný popis konceptu (ať už daného učitelem nebo vytvořeného systémem).

Úlohu empirického učení se konceptům s učitelem (tedy úlohu vytvořit na základě dat klasifikační model) můžeme formálně definovat následujícím způsobem (Berka, 2003):

Předpokládejme, že analyzovaná data jsou uložena v tabulce $\boldsymbol{D}_{\mathrm{TR}}$, tvořené n řádky a m sloupci

$$\boldsymbol{D}_{\mathrm{TR}} = \begin{bmatrix} x_{11} & x_{12} & \ldots\ldots & x_{1m} & y_1 \\ x_{21} & x_{22} & \ldots\ldots & x_{2m} & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \ldots\ldots & x_{nm} & y_n \end{bmatrix}.$$

Řádky tabulky reprezentují sledované *objekty*. Někdy se místo termínu objekt používají termíny záznam (v databázi), příklad, případ, pozorování apod., *i*-tý objekt je tedy řádek $\boldsymbol{x}_i$.

Sloupce datové tabulky odpovídají *atributům*. Podobně jako v případě objektů i zde se používají další termíny – veličina, proměnná, znak. V tuto chvíli nebudeme rozlišovat, zda jde o atributy *kategoriální* (symbolické, diskrétní) nebo atributy *numerické* (spojité). Tato informace bude důležitá až pro jednotlivé typy metod.

U klasifikačních úloh předpokládáme, že existuje atribut, který obsahuje informaci o zařazení objektů do tříd (v případě klasifikace v užším smyslu), nebo který obsahuje predikovanou hodnotu (v případě predikce). Říkejme tomuto atributu *cílový*, ostatním, tj. necílovým, atributům budeme říkat *vstupní* atributy. Opět můžeme v literatuře nalézt řadu dalších názvů: cílovému atributu se někdy říká závislá proměnná, závislá veličina nebo vysvětlovaná veličina, vstupním atributům se také říká nezávislé proměnné, nezávislé veličiny nebo vysvětlující veličiny.

Obsahuje-li datová tabulka cílový atribut, máme data vhodná pro použití některé metody učení s učitelem. Cílem těchto metod je na základě dat tvořených hodnotami vstupních atributů i cílového atributu odvodit znalosti použitelné pro klasifikaci nových objektů. Datům používaným k tomuto účelu se obvykle říká *trénovací data* (trénovací příklady). Příslušnou datovou tabulku budeme značit $\boldsymbol{D}_{\mathrm{TR}}$.

Klasifikační úlohu můžeme chápat jako úlohu nalézt takové *znalosti* (reprezentované rozhodovací funkcí f), které by umožňovaly k hodnotám vstupních atributů $\boldsymbol{x}$ nějakého objektu přiřadit vhodnou hodnotu cílového atributu y:

$$f: \boldsymbol{x} \rightarrow y.$$

Rozhodovací funkci f přitom chápeme v dosti širokém významu. Je-li klasifikace založena na algoritmu používajícím rozhodovací stromy, je tato funkce (a tedy i hledané znalosti) reprezentována jedním konkrétním rozhodovacím stromem. Pokud je klasifikace založena na neuronové síti, pak tuto funkci (a tedy i hledané znalosti) reprezentuje topologie konkrétní sítě a váhy vazeb mezi neurony.

V průběhu klasifikace se tedy pro hodnoty vstupních atributů $\boldsymbol{x}$ nějakého objektu odvodí hodnota cílového atributu. Označme tuto odvozenou hodnotu $\hat{y}$:

$$\hat{y} = f(\boldsymbol{x}).$$

Odvozená hodnota $\hat{y}$ se pro objekty z trénovacích dat může lišit od skutečné hodnoty y. Můžeme tedy pro každý objekt $\boldsymbol{o}_i \in \boldsymbol{D}_{\mathrm{TR}}$ vyčíslit *chybu klasifikace* $Q_f(\boldsymbol{o}_i, \hat{y}_i)$. V případě numerického cílového atributu může být touto chybou například čtverec rozdílu skutečné a odvozené hodnoty cílového atributu

$$Q_f(\boldsymbol{o}_i, \hat{y}_i) = (y_i - \hat{y}_i)^2,$$

v případě kategoriálního cílového atributu může být touto chybou informace o tom, zda se odvozená a skutečná hodnota vzájemně liší, jinak řečeno, zda jsme se dopustili (hodnota 1) nebo nedopustili (hodnota 0) chyby při klasifikování konkrétního objektu $\boldsymbol{o}_i$:

$$Q_f(\boldsymbol{o}_i, \hat{y}_i) = \begin{cases} 1 & \text{pro } y_i \neq \hat{y}_i \\ 0 & \text{pro } y_i = \hat{y}_i \end{cases}.$$

Pro celou trénovací množinu $\boldsymbol{D}_{\mathrm{TR}}$ pak můžeme vyčíslit souhrnnou chybu $\mathrm{Err}(f, \boldsymbol{D}_{\mathrm{TR}})$, například jako střední chybu

$$\mathrm{Err}(f, \boldsymbol{D}_{\mathrm{TR}}) = \frac{1}{n}\sum_{i=1}^{n} Q_f(\boldsymbol{o}_i, \hat{y}_i) .$$

Je-li, tedy cílový atribut kategoriální (provádíme klasifikaci), střední chyba $\mathrm{Err}(f, \boldsymbol{D}_{\mathrm{TR}})$ vyjadřuje podíl špatně zařazených příkladů v rámci celé trénovací množiny. Cílem učení je pak nalézt takové znalosti f^*, které by minimalizovaly tuto chybu

$$\mathrm{Err}(f^*, \boldsymbol{D}_{\mathrm{TR}}) = \min_{f} \mathrm{Err}(f, \boldsymbol{D}_{\mathrm{TR}}) .$$

Empirické metody učení vycházejí z předpokladu, že jednotlivé objekty (příklady, pozorování) lze popsat pomocí charakteristik takových, že objekty patřící k témuž konceptu mají podobné charakteristiky (tyto metody bývají někdy nazývány *učení na základě podobnosti* (similarity-based learning). V této souvislosti je třeba upozornit na skutečnost, že nalézt vhodné charakteristiky splňující uvedený předpoklad zdaleka není triviální záležitostí a vyžaduje značné znalosti o konkrétní úloze i datech, která jsou k dispozici pro analýzy.

Pokud jsou objekty popsány hodnotami atributů, lze je reprezentovat jako body v mnohorozměrném prostoru, jehož dimenze je dána počtem těchto atributů. Učení na základě podobnosti pak vychází z představy, že objekty představující příklady téhož konceptu vytvářejí *shluky* v tomto prostoru. Cílem učení je tedy nalézt vhodné rozdělení příkladů do shluků a každý shluk vhodně reprezentovat. K hledání shluků můžeme přistoupit dvěma základními způsoby:

1. Učení můžeme chápat jako prohledávání prostoru všech shluků (modelů). Tento prostor můžeme uspořádat podle obecnosti modelů. Nejobecnější model odpovídá situaci, kdy všechna data tvoří jeden shluk; tento model bude všechny příklady klasifikovat do majoritní třídy. Nejspeciálnější model odpovídá situaci, kdy každý příklad tvoří samostatný shluk; takový model bude obvykle mít nulovou schopnost generalizace a nebude tedy vhodný pro klasifikaci nových příkladů.

2. Učení můžeme chápat jako aproximaci. V tomto pojetí vycházíme z toho, že jednotlivé shluky lze od sebe oddělit hranicí, kterou můžeme popsat jako funkci hodnot jednotlivých atributů. Při učení se tedy snažíme na základě příkladů jednotlivých tříd nalézt tuto funkci.

10.3 Vybrané adaptivní přístupy

V klasickém pojetí univerzitních experimentů měla úloha dobývání znalostí podobu relativně jednoduché (dávkové) analýzy relativně jednoduchých (a malých) dat. Je tedy od začátku k dispozici celý datový soubor určený pro vytvoření modelu (tzv. trénovacích dat) a tento datový soubor má podobu jedné datové matice obsahující několik málo tisíc řádků (objektů) a několik málo desítek sloupců (atributů). Reálné úlohy dobývání znalostí ale přinášejí nové výzkumné problémy, se kterými je třeba se vyrovnat. V tomto

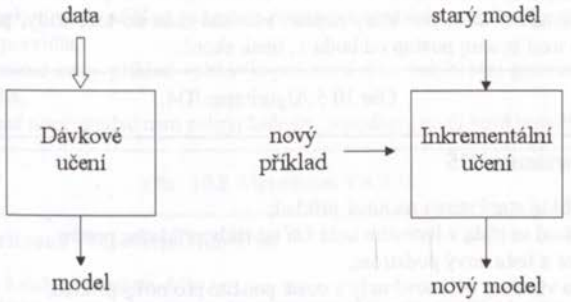
oddílu se zaměříme na situace, kdy je třeba znalosti průběžně modifikovat a aktualizovat, neboli adaptovat.

10.3.1 Inkrementální učení

Při inkrementálním učení lze, na rozdíl od dávkového učení, dosud získané znalosti průběžně modifikovat na základě nových příkladů. Na vstupu inkrementálního algoritmu je původní model a nový příklad, výstupem se stává aktualizovaný model. Výsledek tohoto postupu učení by se ale neměl lišit od dávkového učení, kdy máme k dispozici všechna data na počátku učení.

Některé algoritmy (např. učení založené na instancích) jsou inkrementální ze své podstaty, jiné algoritmy (např. tvorba rozhodovacích stromů) je třeba podstatně modifikovat.

Zajímavou kategorizaci metod inkrementálního učení lze nalézt v práci (Maloof, Michalski, 2004). Metody (a systémy) inkrementálního učení jsou zde posuzovány ze dvou hledisek. Prvním hlediskem je počet příkladů, které systém uchovává v paměti; lze uchovávat všechny příklady (úplná paměť příkladů), žádné příklady (žádná paměť příkladů) nebo jen některé příklady (částečná paměť příkladů). Druhým hlediskem je počet uchovávaných popisů konceptů; opět může systém uchovávat všechny popisy a měnit je jen na základě nových příkladů (úplná paměť konceptů), systém může uchovávat koncepty, ale nemusí je měnit jen jako reakci na nové příklady (částečná paměť konceptů), nebo systém neuchovává žádné koncepty, což je typické pro učení založené na instancích (žádná paměť konceptů).



Obr. 10.3 Dávkové vs. inkrementální učení.

10.3.1.1 Rozhodovací stromy

Při tvorbě rozhodovacího stromu se postupuje metodou „rozděl a panuj" (divide and concquer). Trénovací data se postupně rozdělují na menší a menší podmnožiny (uzly stromu) tak, aby v těchto podmnožinách převládaly příklady jedné třídy. Na počátku tvoří celá trénovací data jednu množinu, na konci máme podmnožiny tvořené příklady téže třídy. Tento postup bývá často nazýván „top down induction of decision trees" (TDIDT). Obecné schéma dnes již klasického algoritmu ID3 (Quinlan, 1986) vidíme na obr. 10.4.

Algoritmus ID3

1. zvol jeden atribut jako kořen dílčího stromu,
2. rozděl data v tomto uzlu na podmnožiny podle hodnot zvoleného atributu a přidej uzel pro každou podmnožinu,
3. existuje-li uzel, pro který nepatří všechna data do téže třídy, pro tento uzel opakuj postup od bodu 1, jinak skonči.

Obr. 10.4 Algoritmus ID3.

Standardní podoba algoritmu pro tvorbu rozhodovacích stromů je tedy dávková. Existují ale i inkrementální varianty, např. systémy ID4 (Schlimmer, Fisher, 1986) a ID5 (Utgoff a kol., 1997). Obecné schéma algoritmu ID4 vidíme na obr 10.5 a obecné schéma algoritmu ID5 je znázorněno na obr. 10.6. Z hlediska uvedené kategorizace jde v případě ID4 (Maloof, Michalski, 2002) o systém používající úplnou paměť konceptů a žádnou paměť příkladů a v případě ID5 o systém používající úplnou paměť konceptů i úplnou paměť příkladů.

Algoritmus ID4

1. na základě nového příkladu nalezni nejvhodnější atribut pro větvení,
2. není-li nalezený atribut použit pro větvení,
   2.1 odstraň celý podstrom,
   2.2 použij vybraný atribut pro větvení (rozděl data na podmnožiny a přidej uzly do stromu),
3. existuje-li uzel, pro který nepatří všechna data do téže třídy, pak pro tento uzel opakuj postup od bodu 1, jinak skonči.

Obr.10.5 Algoritmus ID4.

Algoritmus ID5

1. aplikuj starý strom na nový příklad,
2. pokud se třída v listovém uzlu liší od třídy příkladu, potom vytvoř z listu nový podstrom,
3. pro všechny nelistové uzly v cestě použité pro nový příklad,
   3.1 pokud atribut použitý pro větvení není po zahrnutí nového příkladu nejlepší,
      3.1.1 použij nejlepší atribut,
      3.1.2 rekurzivně transformuj celý podstrom.

Obr. 10.6 Algoritmus ID5.

10.3.1.2 Rozhodovací pravidla

K nejznámějším algoritmům pro tvorbu pravidel patří algoritmy pokrývání množin pracující metodou „odděl a panuj" (separate and conquer). Při pokrývání množin totiž hledáme pravidla, která popisují příklady téže třídy a oddělují je od příkladů třídy jiné. Základní podobu algoritmu pokrývání množin pro dvě třídy ukazuje obrázek 10.7.

V rámci tohoto obecného postupu jsou dvě základní možnosti jak (v kroku 1) najít jedno pravidlo:

- při pokrývání „zdola nahoru", např. AQ (Michalski, 1969) – se vyjde od nejspeciálnějšího pravidla, které má v předpokladu konjunkci hodnot všech vstupních atributů (a pokrývá tedy jediný příklad) a postupně se odstraňují podmínky z předpokladu,
- při pokrývání „shora dolů", např. CN2 (Clark, Nibblet, 1989) – se vyjde od nejobecnějšího pravidla, které nemá žádnou podmínku v předpokladu (a pokrývá tedy celou trénovací množinu) a postupně se přidávají podmínky do předpokladu.

Algoritmus pokrývání množin

1. najdi pravidlo, které pokrývá nějaké pozitivní příklady a žádný negativní,
2. odstraň pokryté příklady z trénovací množiny,
3. pokud v trénovací množině zbývají nějaké nepokryté pozitivní příklady, vrať se k bodu 1, jinak skonči.

Obr. 10.7 Obecné schéma algoritmu pokrývání množin.

Vedle základních, dávkových algoritmů existují i inkrementální varianty. Příkladem mohou být systémy YAILS (Torgo, 1992) a PRIMEROSE-INC (Tsumoto, Tanaka, 1997).

Algoritmus YAILS

1. pro každé existující pravidlo,
   1.1 pokud nový příklad vyhovuje předpokladu i závěru pak nedělej nic,
   1.2 pokud nový příklad vyhovuje pouze předpokladu, pak hledej specializaci pravidla,
   1.3 pokud nový příklad vyhovuje pouze závěru, pak hledej generalizaci pravidla,
2. Pokud nový příklad není pokryt žádným pravidlem, najdi nové pravidlo.

Obr. 10.8 Algoritmus YAILS.

Algoritmus PRIMEROSE-INC

1. pro každou kategorii A(v),
pokud klesne její pokrytí po vstupu nového příkladu, zařaď A(v) do seznamu K, jinak zařaď A(v) do seznamu R,
2. pro každou kategorii A(v) ze seznamu K,
najdi v seznamu „Vyhov" pravidlo, které obsahuje A(v) a které nyní nevyhovuje požadavkům na spolehlivost a pokrytí a přesuň je do seznamu „Nevyhov",
3. pro každou kategorii A(v) ze seznamu R najdi v seznamu „Nevyhov" pravidlo, které obsahuje A(v),
   3.1 pokud toto pravidlo nyní vyhovuje požadavkům na spolehlivost a pokrytí, přesuň je do seznamu „Vyhov", jinak hledej jeho vyhovující specializaci,
4. pokud ostatní pravidla z „Nevyhov" nyní vyhovují, přesuň je do „Vyhov".

Obr. 10.9 Algoritmus PRIMEROSE-INC.

10.3.1.3 Neuronové sítě

Umělé neuronové sítě vycházejí z analogie s lidským mozkem. Podobně jako mozek jsou tvořeny množstvím navzájem propojených elementů – neuronů. V umělých neuronových sítích je neuron chápán jako buňka, která přijímá podněty od jiných neuronů, které jsou k ní připojeny „na vstupu". Pokud souhrnný účinek těchto vstupních podnětů překročí určitý práh, neuron se aktivuje a sám začne svým výstupem působit na další neurony. Složitější umělé neuronové sítě bývají tvořeny množstvím různě navzájem propojených neuronů. Pro klasifikační, popř. predikční, úlohy se obvykle používá vícevrstvá síť (vícevrstvý perceptron, multi-layer perceptron) složená z vrstev neuronů, kde v rámci jedné vrstvy nejsou mezi neurony žádné vazby, ale neuron z jedné vrstvy je propojen se všemi neurony vrstvy sousední. Při učení se v neuronových sítích je typickým postupem učení jako aproximace funkce: známe topologii sítě, což je implicitní vyjádření typu funkční závislosti mezi vstupy a výstupy sítě, a hledáme „jen" parametry modelu. Pro učení tohoto typu sítí se používá algoritmus zpětného šíření chyby (error backpropagation), kdy se postupně (od výstupní vrstvy směrem ke vstupní vrstvě) modifikují váhy vazeb mezi neurony s cílem minimalizovat chybu klasifikace pro celou trénovací množinu. Příklady v trénovací množině přitom můžeme zpracovávat dávkově (váhy se modifikují až po zpracování celého souboru příkladů) nebo inkrementálně (váhy se modifikují po každém příkladu). Ilustraci obou přístupů při učení jednoduchého perceptronu vidíme na obrázcích 10.10 (dávkový přístup) a 10.11 (inkrementální přístup).

Dávkový algoritmus

1. náhodně inicializuj váhy,
2. dokud není splněno kritérium pro zastavení,
   2.1 zpracuj všechny trénovací příklady $[\boldsymbol{x}_i, y_i]$,
   2.2 aktualizuj váhy

$$\Delta \boldsymbol{w} = \eta \sum_i (y_i - \hat{y}_i)\boldsymbol{x}_i .$$

Obr. 10.10 Dávková verze algoritmu pro učení neuronových sítí.

Inkrementální algoritmus

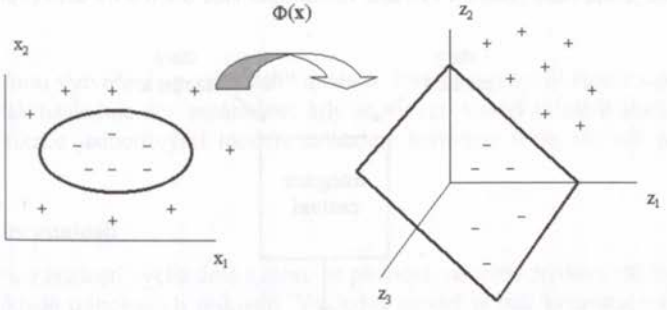
1 náhodně inicializuj váhy,
2. dokud není splněno kritérium pro zastavení,
   2.1 pro každý trénovací příklad,
      2.1.1 zpracuj příklad $[\boldsymbol{x}_i, y_i]$,
      2.1.2 aktualizuj váhy,

$$\Delta \boldsymbol{w}_i = \eta\,(y_i - \hat{y}_i)\,\boldsymbol{x}_i .$$

Obr. 10.11 Inkrementální verze algoritmu pro učení neuronových sítí.

10.3.1.4 SVM

Hlavní myšlenkou metody SVM je převést pomocí vhodné datové transformace úlohu klasifikace do tříd, které nejsou lineárně separabilní, na úlohu klasifikace do tříd lineárně separabilních (viz obr. 10.12). Místo v prostoru původních atributů se pak pohybujeme v prostoru transformovaných atributů (příznaků). Datová transformace sice představuje nárůst dimenze (oproti původním atributům je počet příznaků větší), řešíme ale jednodušší úlohu. V tomto novém prostoru hledáme rozdělující nadrovinu, která má největší odstup od transformovaných příkladů z trénovací množiny (tzv. maximal margin hyperplane). Rozhodující pro nalezení rozdělující nadroviny jsou samozřejmě příklady, které leží nejblíže hledané hranici mezi třídami. Tyto takzvané „podpůrné" vektory (support vectors) daly jméno celé metodě.



Obr. 10.12 Princip metody SVM.

Učení (hledání rozdělující nadroviny) má podobu úlohy kvadratického programování. Základní podoba algoritmu je dávková, pro učení se použijí všechny trénovací příklady, např. (Cortes, Vapnik, 1995). V inkrementální verzi algoritmu se pro učení použijí staré podpůrné vektory (vážené počtem starých příkladů na jeden podpůrný vektor) a nová data (Ruping, 2001).

10.3.1.5 Učení založené na instancích

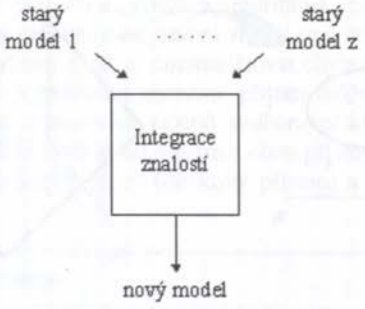
V případě učení založeném na instancích jsou koncepty (třídy) reprezentovány svými typickými představiteli. Tyto představitele (v kontextu strojového učení nazývané instance) můžeme získat tak, že si zapamatujeme (uložíme do báze případů) všechny trénovací příklady. Takto, bez prohledávání, postupuje například algoritmus IB1 (Aha a kol., 1991). Není ale třeba ukládat vše, stačí uložit pouze ty příklady, které nejsou zatím správně klasifikovány. Tento postup, který odpovídá jednoduchému heuristickému prohledávání shora dolů (špatně klasifikovaný příklad vytvoří zárodek nového shluku), nalezneme v algoritmech IB2 a IB3, rovněž (Aha a kol., 1991). V tomto případě jde o postup, který je inkrementální ze své podstaty a jde o přístup používající úplnou paměť konceptů a úplnou (v případě IB1) nebo částečnou (v případě IB2 a IB3) paměť příkladů.

10.3.2 Integrování znalostí

Pojmem integrování znalostí chápeme proces kombinace znalostí získaných z různých zdrojů, popř. různými způsoby. K integrování znalostí může (v rámci metod dobývání znalostí a strojového učení) docházet na úrovni reprezentace nebo na úrovni usuzování.

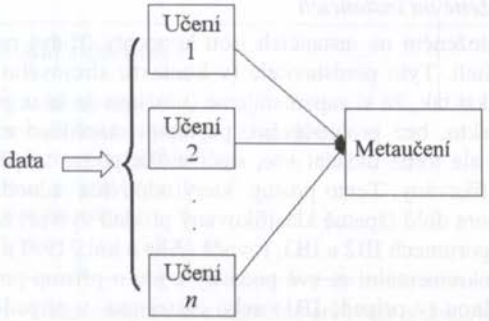
Na úrovni reprezentace se může jednat o
- kombinování znalostí získaných z různých souborů trénovacích dat,
- kombinování doménových znalostí a znalostí získaných z dat do jednoho znalostního modelu.

Integrování znalostí je tedy v tomto smyslu úzce svázáno s inkrementálním učením (popsaném v předcházejícím oddílu) nebo s revizí znalostí (zmíněném v následujícím výkladu). Obecněji patří problematika integrování znalostí do oblasti znalostního inženýrství, kde jedna báze znalostí vznikne kombinací více dílčích modelů (obr. 10.13).



Obr. 10.13 Integrování na úrovni reprezentace.

Na úrovni usuzování jde o různé způsoby kombinování modelů. Cílem zde není získat jeden model, ale jedno rozhodnutí. Toto pojetí bývá často označováno jako metaučení (meta-learning) nebo multistrategy learning. Obecné schéma kombinování modelů vidíme na obr. 10.14.



Obr. 10.14 Kombinování modelů.

Porovnání různých přístupů ke kombinování modelů můžeme nalézt v článcích (Bauer, Kohavi, 1999) nebo (Dietrich, 2000). K základním přístupům patří bagging, boosting a stacking.

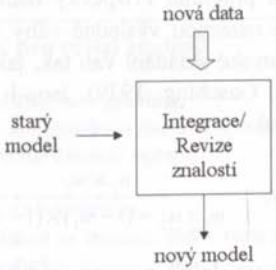
Bagging
Učení je založeno na vytvoření „paralelních" modelů na základě různých podmnožin trénovacích dat (náhodný výběr s opakováním). Při klasifikaci nového příkladu volíme rovné hlasování modelů.

Boosting
Učení má podobu vytvoření sekvence modelů. Každý model se zaměřuje na ty příklady, které se předcházejícími modely nepodařilo klasifikovat (vážení příkladů). Přitom se použije týž algoritmus pro učení. Při klasifikaci nového příkladu volíme vážené hlasování modelů.

Stacking
Učení má podobu vytvoření „paralelních" modelů. Přitom se použijí různé algoritmy na stejná data. Pak následuje tzv. *metaučení*, kdy se vytváří model (klasifikátor), který na základě klasifikace jednotlivými modely odhaduje správnou třídu, do níž patří nový příklad.

10.3.3 Revize znalostí

V případě revize znalostí vycházíme z toho, že původní, neúplné znalosti můžeme modifikovat na základě trénovacích příkladů. Výsledný model je pak konzistentní s původními znalostmi i s novými příklady.



Obr. 10.15 Úloha revize znalostí.

10.3.3.1 Induktivní logické programování

Oblastí, která uvedeným způsobem přistupuje k úloze strojového učení, je induktivní logické programování (ILP). Úlohu ILP můžeme definovat tímto způsobem: Je dána počáteční znalost B a trénovací příklady pozitivní E^+ a negativní E^- (z hlediska jedné třídy). Cílem je nalézt nové znalosti H tak, aby:
- z H a B byly odvoditelné všechny pozitivní příklady,
- z H a B nebyl odvoditelný žádný negativní příklad,
- H a B byly konzistentní.

K průkopníkům oblasti ILP patří S. Muggleton (Muggleton, 1992). Dnes už existuje celá řada systémů, které implementují princip ILP ve spojení s „klasickými" algoritmy strojového učení (tab. 10.1).

Tabulka 10.1 Systémy ILP

Typ algoritmu	ILP
rozhodovací stromy	TILDE
klasifikační a regresní stromy	SCART
rozhodovací pravidla	FOIL, PROGOL
asociační pravidla	Claudien, Warmr
nejbližší soused	RIBL

10.3.3.2 Revize pravidel

Myšlenku revize znalostí nalezneme i u klasických algoritmů strojového učení používajících reprezentaci příkladů založenou na hodnotách atributů. Příkladem nám budiž revize pravidel.

Podívejme se nejprve na algoritmus KEX, viz např. (Berka, Iványek, 1994), který vytváří pravidla tvaru

$$Ant \Rightarrow Class\ (w),$$

kde *Ant* je kombinace kategorií vstupních atributů, *Class* značí kombinaci definující pozitivní příklady (tedy $Class = C(+)$) a w z intervalu [0, 1] je váha vyjadřující neurčitost pravidla.

V průběhu klasifikace se pro nový objekt popsaný hodnotami vstupních atributů naleznou všechna aplikovatelná pravidla. Příspěvky těchto pravidel se složí pomocí kombinační funkce $\oplus$, a tak se naleznou výsledné váhy všech cílů. Jako kombinační funkce se používá pseudobayesovské skládání vah tak, jak bylo zavedeno v expertním systému PROSPECTOR (Duda, Gasching, 1979). Jsou-li w_1, w_2 váhy dvou pravidel, spočítá se jejich kombinace $w^{\oplus}$ jako

$$w_1 \oplus w_2 = \frac{w_1 \times w_2}{w_1 \times w_2 + (1-w_1)\times(1-w_2)}.$$

Bázi pravidel vytváříme „shora dolů" postupným přidáváním nových (speciálnějších) pravidel ve chvíli, kdy báze přestane být konzistentní s trénovacími daty reprezentovanými souborem všech implikací, které mají zadaný cíl ve svém závěru. Na počátku obsahuje báze pravidel tzv. prázdný vztah, který odpovídá rozdělení tříd v trénovacích datech. Do báze pravidel se pak postupně zařazují jen ty implikace, které nejsou odvoditelné z již získaných kratších pravidel. Přesněji řečeno, zařadí se jen ty implikace, jejichž spolehlivost (počítaná jako $P(Class|Ant)$) se výrazně liší od váhy závěru této implikace počítané s využitím uvedené kombinační funkce. V základní variantě jde o algoritmus, který pracuje v dávkovém režimu (obr. 10.16).

Algoritmus KEX dávková verze

nechť *CAT* je seznam kategorií $A(v)$,
nechť *IMPL* je seznam implikací $A(v) \Rightarrow Class$
(oba seznamy jsou uspořádány sestupně dle četnosti $n(A(v))$),
nechť báze pravidel *KB* obsahuje pouze prázdný vztah $\varnothing \Rightarrow Class\ (w)$,

dokud *IMPL* není prázdný seznam,
1. vezmi první implikaci ze seznamu *IMPL* (označ ji $Ant \Rightarrow Class$),
2. spočítej váhu $w^{\oplus}(Ant)$,
3. pokud se $P(Class|Ant)$ na základě χ^2 testu liší od $w^{\oplus}(Ant)$, potom,
 3.1 přidej do *KB* pravidlo $Ant \Rightarrow Class\ (w)$, kde
 $$w \oplus w^{\oplus}(Ant) = P(Class|Ant),$$
 3.2 prodluž *Ant* o všechny vhodné kategorie $A(v)$ z *CAT* a
 zařaď $Ant \wedge A(v) \Rightarrow Class$ podle četnosti do seznamu *IMPL*,
4. odstraň $Ant \Rightarrow Class$ ze seznamu *IMPL*.

Obr. 10.16 KEX – dávková verze.

Varianta algoritmu pro revizi znalostí porovnává implikace vytvořené pro nová trénovací data a dříve vytvořená pravidla. Pokud nějaká implikace $Ant \Rightarrow Class$ bude zařazena jako nové pravidlo, může to ovlivnit taková již existující pravidla $Comb_r \Rightarrow Class\ (w_r)$, pro která je kombinace $Comb_r$ prodloužením (nadkombinací) kombinace *Ant*. Tato pravidla je tedy třeba rovněž otestovat a buď je z báze pravidel odstranit (jsou-li nyní zbytečná), nebo modifikovat jejich váhu. Schéma algoritmu vidíme na obr. 10.17.

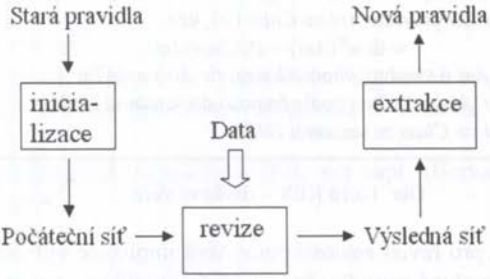
Algoritmus KEX pro revizi znalostí

nechť KB je již existující báze pravidel,
nechť IMPL je seznam implikací $Ant \Rightarrow Class$ vytvořených z nových dat uspořádaný sestupně dle četnosti $n(Ant)$,

dokud IMPL není prázdný seznam,
1. vezmi první implikaci ze seznamu *IMPL* (označ ji $Ant \Rightarrow Class$),
2. spočítej váhu $cw(Ant)$,
3. pokud se $P(Class|Ant)$ na základě testu χ^2 liší od $cw(Ant)$, potom,
 3.1 přidej nebo přepiš do *KB* pravidlo $Ant \Rightarrow Class(w)$,
 3.2 pro každé pravidlo $Comb_r \Rightarrow Class(w_r)$ z *KB* takové, že *Ant*
je podkombinací kombinace $Comb_r$,
 3.2.1 spočítej váhu $cw(Comb_r)$,
 3.2.2 pokud se $P(Class|Comb_r)$ signifikantně liší od$cw(Comb_r)$, potom
modifikuj váhu w_r, jinak ostraň pravidlo $Comb_r \Rightarrow Class(w_r)$ z *KB*,
4. odstraň $Ant \Rightarrow Class$ ze seznamu *IMPL*.

Obr. 10.17 KEX pro revizi znalostí.

Jsou samozřejmě i jiné přístupy k revizi pravidel. Obrázek 10.18 ukazuje základní myšlenku revize pravidel s využitím neuronových sítí tak, jak je implementována v celé řadě systémů. Původní pravidla jsou použita pro konfigurování neuronové sítě, síť je učena s využitím nových příkladů a naučená síť je pak použita pro extrakci nových pravidel. U systému KBANN (Towell a Shavlik, 1993), popř. v práci (Berka a Sláma, 1998), tento postup nevede ke změně struktury pravidel (neuronová síť je zafixována tak, aby se modifikovaly pouze váhy vazeb, které odpovídají původním pravidlům). V systému RAPTURE (Mahoney, Mooney, 1994) se modifikují váhy vazeb v plně propojené neuronové síti, což může vést k extrakci nových pravidel.



Obr. 10.18 Schéma revize pravidel pomocí neuronových sítí

10.3.4 Posun konceptu

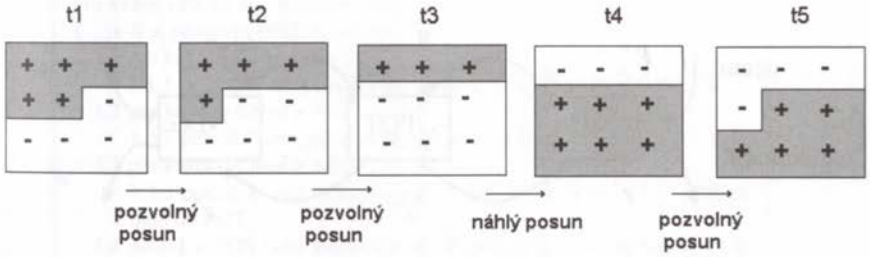
Zatím jsme v našich úvahách předpokládali, že všechna data, která používáme pro inkrementální učení nebo pro revizi znalostí, odpovídají stejným skrytým znalostem, jež chceme objevit. Že se tedy vztah mezi vstupními atributy na straně jedné a výstupním (cílovým) atributem na straně druhé řídí stále stejnými zákonitostmi a že tedy jsou všechny příklady stejně relevantní v procesu strojového učení. Ne vždy ale tento předpoklad platí; v různých úlohách se může stát, že relevantní jsou pouze nejnovější příklady. V literatuře je tato situace obvykle označována jako posun konceptu (koncept drift) nebo dynamická změna v doméně (dynamically changing domain).

Posun konceptu lze rozpoznat tak, že model vytvořený a otestovaný pro starší data přestává správně klasifikovat či predikovat pro data nová. Jinými indikátory pro posun konceptu mohou být růst složitosti nalezených modelů (např. růst počtu pravidel) nebo změna v charakteristikách nových příkladů (např. změna v rozdělení příkladů k jednotlivým třídám). Problémem je zde ovšem skutečnost, že stejným způsobem se může projevovat i šum v datech. Příliš senzitivní přístup tak bude detekovat posun konceptu i v případě šumu, příliš robustní přístup bude naopak posun konceptu považovat za šum.

Posun konceptu může být pozvolný nebo náhlý. Můžeme například úspěšně klasifikovat klienty banky žádající o úvěr z hlediska jejich bonity na základě znalostí získaných z dat jen do té doby, než se změní předpisy o poskytování úvěrů (náhlý posun konceptu), nebo lze na základě chování návštěvníka webových stránek odhadovat jeho oblasti zájmu, které se ale mohou postupně měnit (pozvolný posun konceptu). Rozdíl mezi těmito situacemi ilustruje obrázek 10.19 uvedený v článku (Nunez a kol., 2007).

Posun konceptu se navíc může cyklicky opakovat; příkladem může být předpověď počasí, která se řídí různými zákonitostmi pro jednotlivá roční období.

O posunu konceptu má smysl mluvit, pokud uvažujeme i faktor času. Sběr dat tedy obvykle probíhá postupně (čemuž odpovídá například pravidelná měsíční, čtvrtletní apod. aktualizace datového skladu), ale mezi jednotlivými datovými sadami může dojít ke změně konceptu. Typickým se tedy stává inkrementální přístup k učení.



Obr. 10.19 Posun konceptu.

S posunem konceptu je možné se na úrovni dat vypořádat v zásadě dvěma způsoby. První možností je vážit jednotlivé příklady tak, že nejnovější (nejaktuálnější) příklady budou mít největší váhu. Jinou možností je pracovat s oknem, které vybere jen určitou (opět nejaktuálnější) sadu příkladů. Mezi oběma způsoby není zásadního rozdílu; vybírání příkladů pomocí okna odpovídá situaci, kdy příklady vážíme vahou 1 (příklad je vybrán) nebo vahou 0 (příklad vybrán není).

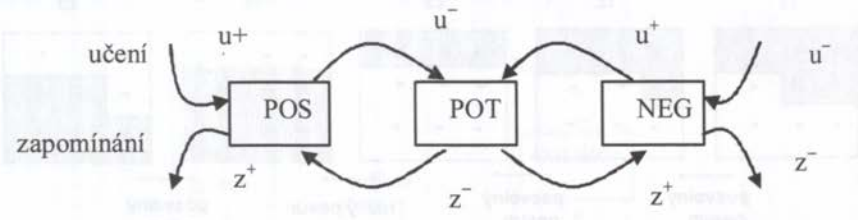
Podle toho, jak vypadá výběr příkladů, rozlišujeme režimy:
- full memory (k dispozici jsou všechny příklady, které byly kdy posbírány),
- no memory (pracuje se pouze s příklady z poslední sady),
- fixed window (příklady se vybírají oknem pevné délky, která nemusí nutně zahrnovat pouze příklady z poslední sady),
- adaptive window (příklady se vybírají oknem, jehož délka se průběžně mění; je-li koncept stabilní, okno se zvětšuje, mění-li se koncept, okno se zmenšuje).

Jinou možností je vážit nikoliv příklady, ale dílčí modely. V této situaci můžeme vytvořit samostatné modely z jednotlivých sad příkladů a tyto modely pak kombinovat některou ze zmíněných metod metaučení. Váha jednotlivých modelů by pak brala v úvahu jejich stáří a tím i relevanci pro aktuální rozhodování.

10.3.4.1 Učení a zapomínání

Učení a zapomínání hraje roli v úlohách, kdy koncepty, jimž je třeba se naučit, se průběžně mění. Vzhledem k tomu, že se průběžně mění koncept, který se chceme naučit, nelze pro učení použít všechny příklady. Na rozdíl od „čistého" inkrementálního učení, kdy považujeme za relevantní všechny příklady, které postupně používáme, zde hrají roli jen příklady nejnovější.

Obr. 10.20 ilustruje přístup založený na použití okna, ve kterém jsou uchovávány jen aktuální příklady. Toto schéma bylo publikováno v článku (Widmer, Kubát, 1996) v souvislosti s učením se pravidlům v systému FLORA, můžeme ho ale chápat jako obecnou ilustraci všech přístupů k učení a zapomínání. Na jedné straně jsou do okna přidávány nové příklady (systém se učí), na druhé straně jsou z okna odstraňovány příklady staré (systém zapomíná). Při učení, popř. zapomínání, se modifikuje ta část znalostí, která pokrývá přidávaný, popř. odebíraný, příklad.



Obr. 10.20 Učení a zapomínání dle článku (Widmer, Kubát, 1996).

K prvním systémům, které byly schopny se vypořádat s posunem konceptu, patřily systémy STAGGER (Schlimmer, Granger, 1986), IB3 (Aha a kol., 1991) a FLORA (Widmer, Kubát, 1996).

STAGGER

Systém STAGGER během učení aktualizuje na základě nové evidence míru postačitelnosti a míru nutnosti hypotéz popisujících příslušný koncept (jde tedy o bayesovský přístup). Důsledkem je průběžné posilování nebo oslabování jednotlivých hypotéz. Systém je citlivý na délku procesu učení; čím déle se systém učil na základě příkladů jednoho konceptu, tím lépe si tento koncept zapamatuje a tím méně je „ochoten" přizpůsobit se konceptu jinému.

IB3

Systém IB3 je systém učení založený na instancích, který inkrementálně přidává do báze případů ty příklady, jež byly chybně klasifikovány stávající bází (učení). Současně ale zjišťuje, jak úspěšně se případy v bázi podílejí na klasifikaci nových příkladů, a případy, u nichž klesá úspěšnost, z báze případů odstraní (zapomínání).

FLORA

Jádrem algoritmu v systému FLORA je hledání pravidel pokrývajících příklady dané třídy metodou generalizace analogicky jako u algoritmu AQ. Pravidla jsou ukládána do tří množin (obr 10.20): množina POS zahrnuje pravidla pokrývající pouze pozitivní příklady, množina NEG obsahuje pravidla pokrývající pouze negativní příklady a množina POT je tvořena pravidly, která pokrývají pozitivní i negativní příklady.

Učení spočívá v tom, že množiny POS, NEG a POT jsou aktualizovány po každém načtení nového příkladu. Je-li například nový pozitivní příklad pokryt hypotézou z NEG, přesune se tato hypotéza do POT (viz obr 10.21). Postup *zapomínání* je analogický postupu učení. Starší příklady jsou zapomínány, což opět vede ke změnám v mno-

žinách POS, NEG a POT. Jsou-li třeba zapomenuty všechny příklady, které odpovídaly nějakému pravidlu z množiny POS, toto pravidlo se odstraní (viz obr. 10.22). Tato základní varianta algoritmu (která předpokládá, že data neobsahují šum) byla později modifikována tak, že pravidla v množinách POS a NEG nemusí pokrývat „pouze" příklady jedné třídy a pokrývají „dostatečnou většinu" příkladů.

Algoritmus FLORA pro učení

Pro každý příklad o_i z trénovací množiny,
1 je-li o_i pozitivní příklad, potom,
1.1 pro každou *Comb* z POS,
1.1.1 pokud *Comb* pokrývá o_i, přiřaď $n^+(Comb) := n^+(Comb) + 1$,
1.2 pro každou *Comb* z POT,
1.2.2 pokud *Comb* pokrývá o_i, přiřaď $n^+(Comb) := n^+(Comb) + 1$,
1.3 pro každou *Comb* z NEG,
1.3.3 pokud *Comb* pokrývá o_i, přiřaď $n^+(Comb) := 1$ a přesuň *Comb* do POT,
1.4 pokud v POS není žádná *Comb*, která pokrývá o_i, přidej do POS novou kombinaci CombN, která pokryje o_i a nebude (kvůli sporu) v souladu s hypotézami v POT a NEG, a přiřaď $n^+(CombN) := 1$,
2. je-li o_i negativní příklad, potom postupuj analogicky...

Obr. 10.21 Algoritmus FLORA – proces učení.

Algoritmus FLORA pro zapomínání

Pro každý příklad o_i z trénovací množiny,
1. je-li o_i pozitivní příklad, potom,
1.1 pro každou *Comb* z POS,
1.1.1 pokud *Comb* pokrývá o_i, přiřaď $n^+(Comb) := n^+(Comb) - 1$,
1.1.2 je-li $n^+(Comb) = 0$, odstraň *Comb* z POS,
1.2 pro každou *Comb* z POT,
1.2.1 pokud *Comb* pokrývá o_i, přiřaď $n^+(Comb) := n^+(Comb) - 1$,
1.2.2 je-li $n^+(Comb) = 0$, přesuň *Comb* do NEG,
2. je-li o_i negativní příklad, potom postupuj analogicky...

Obr. 10.22 Algoritmus FLORA – proces zapomínání.

Přizpůsobení se posunu konceptu je umožněno dynamickou modifikací velikosti okna (počtu příkladů uchovávaných v paměti). Myšlenka této modifikace je poměrně jednoduchá, pokud se koncept nemění, okno má stabilní velikost (aby se hypotézy vytvářely na základě hodně příkladů), pokud se koncept začíná měnit, okno se zmenšuje (zapomínají se příklady, které odpovídají starému konceptu). Jak ale poznat začínající změnu konceptu? Změna konceptu je automaticky indikována tehdy, když klesá podíl počtu pozitivních příkladů pokrytých množinou POS k velikosti množiny POS. Tento pokles znamená, že se nedaří nalézt dostatečně obecné hypotézy pokrývající pozitivní příklady a že se tedy něco děje. Reakcí systému na tuto skutečnost je zmenšení velikosti okna o 10 %. Pokud se zdá být kontext stabilní (uvedený podíl je vysoký), velikost okna

se nemění. Pokud není podíl ani velký, ani malý (systém se zotavuje ze změny konceptu), okno roste tak, že po načtení nového příkladu se nejstarší příklad nezapomene.

10.3.4.2 Koncepty závislé na kontextu

Někteří autoři přisuzují změnu konceptu, které jsme se věnovali v předcházejícím textu, různému kontextu, v němž řešíme úlohu dobývání znalostí. Kontextem v jejich pojetí může být (Matwin, Kubát, 1996):

- situace, ve které získáváme data,
- relevantní atributy, které nejsou v současnosti dostupné,
- atributy, jež samy o sobě nepřispívají ke klasifikaci, ale které zlepšují výsledky klasifikace v kombinaci s jinými atributy.

V dávkovém přístupu jsou skryté, stabilní kontexty hledány v celém datovém souboru. Kontextově závislé koncepty pak mohou být použity jako část násobného modelu (v rámci kombinace modelů), lokální koncepty mohou být verifikovány expertem a „poskládány" do globálního konceptu, identifikované kontexty mohou sloužit při hledání nových atributů použitelných pro explicitní vyjádření kontextu v trénovacích datech. V dávkovém učení je základním požadavkem, aby znalost získaná v jednom kontextu byla použitelná i v kontextu jiném, např. (Turney, 1996).

Příkladem dávkového přístupu k učení kontextově závislým konceptům může být systém SPLICE (Harries a kol., 1998). Tento systém provádí takzvané kontextové shlukování s cílem identifikovat stabilní skryté kontexty a s nimi spojené lokálně stabilní koncepty. Využívá přitom kombinování modelů vytvořených nějakým klasickým algoritmem, např. algoritmem C4.5 pro tvorbu rozhodovacích stromů. Systém SPLICE postupuje ve třech krocích:

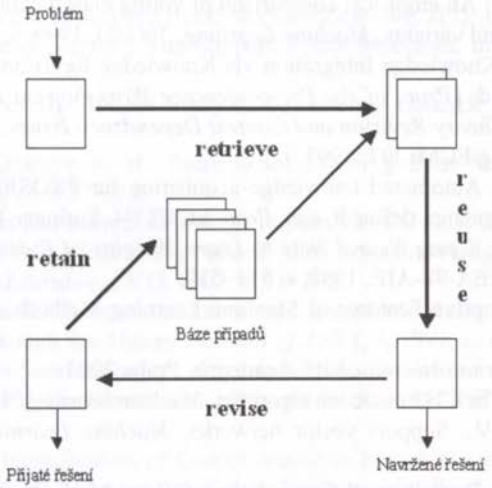
1. rozkladem datového souboru,
2. kontextovým shlukováním,
3. učením lokálního konceptu.

V prvním kroku se nejprve vytvoří (pomocí C4.5) počáteční koncept z celých dat. Ve vytvořeném rozhodovacím stromu můžeme rozpoznat časové intervaly, které přispěly ke klasifikaci (ve stromu se jako větvící atribut objeví časově závislý atribut definující tyto intervaly). Každý takovýto interval je chápán jako lokální kontext definovaný pro určitou podmnožinu trénovacích příkladů; odpovídající lokální koncept je pak popsán tou částí stromu, který odpovídá příslušné podmnožině. Ve druhém kroku se provádí kontextové shlukování intervalů (podmnožin) trénovacích dat. Míra podobnosti podmnožin je dána přesností klasifikace dílčího modelu na těchto podmnožinách (porovnávají se všechny podmnožiny a všechny dílčí modely). Ve třetím kroku se použije učící se algoritmus C4.5 pro učení lokálního konceptu na podmnožině dat, identifikované v předcházejícím kroku. V tomto kroku se ale již nepracuje s atributem použitým pro identifikaci podmnožin.

10.3.5 Analogie a adaptace

Základní myšlenkou usuzování na základě analogie je použít řešení, jež se v minulosti osvědčilo, na podobné úloze, kterou řešíme nyní. Tento princip našel své vyjádření ve statistice (klasifikace dle nejbližšího souseda), případovém usuzování i strojovém učení (tzv. učení založeném na instancích, instance-based learning).

Případové usuzování (Case-Based Reasoning, CBR) pracuje způsobem naznačeným na obr. 10.23 (Watson, Marir, 1994):
- retrieve (najdi nejpodobnější případy),
- reuse (použij navržené řešení),
- revise (popřípadě reviduj navržené řešení),
- retain (uchovej nové řešení v bázi případů = učení).



Obr. 10.23 Případové usuzování.

Pro algoritmy učení založené na instancích je typické, že ve fázi učení se neprovádí generalizace z příkladů. Model je tvořen bází případů (vybraných příkladů z trénovacích dat) a má tedy lokální povahu. Tato skutečnost usnadňuje jak inkrementální postup, tak možnost adaptace (např. na změnu konceptu). Při usuzování se pak v bázi případů hledá nejpodobnější případ, dříve použité řešení někdy musí být adaptováno na novou situaci. K nejznámějším algoritmům učení založeným na instancích patří algoritmy IB1, IB2 a IB3, o nichž jsme se zmínili v předcházejícím výkladu.

10.4 Závěr

Cílem této kapitoly bylo ukázat, jak lze klasické algoritmy strojového učení a dobývání znalostí obohatit o prvky adaptivity. Tato adaptivita je často jedinou možností, jak řešit reálné úlohy dobývání znalostí, kde data mají podobu datových proudů (data streams), jako je tomu například u dat z oblasti telekomunikací, počítačových sítí nebo internetu, nebo kde data jsou příliš rozsáhlá, jsou k dispozici pro analýzu jen postupně a jejich charakteristiky se průběžně mění. Problematice adaptivity je tedy věnována v komunitě dobývání znalostí stále větší pozornost.

Literatura

Aha D., Kibler D., Albert M. K.: Instance-based Learning Algorithms, *Machine Learning*, 6(1), 1991, s. 37–66.

Anguita D.: *Smart Adaptive Systems: State of the Art and Future Directions of Research.* European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems EUNITE 2001, Tenerife 2001.

Bauer E., Kohavi R.: An empirical comparison of voting classification algorithms: Bagging, boosting, and variants, *Machine Learning*, 36(1/2), 1999, s. 105–139.

Berka P.: Towards Knowledge Integration via Knowledge Revision. In: Nakhaeizadeh, Bruha, Taylor, eds: *Proc. of the Pre-conference Workshop on Dynamically Changing Domains: Theory Revision and Context Dependence Issues.* European Conf. on Machine Learning ECML'97, 1997.

Berka P., Ivánek J.: Automated knowledge acquisition for PROSPECTOR-like expert systems. In: Bergadano, deRaedt, eds: *Proc. ECML'94*. Springer 1994, s. 339–342.

Berka P., Sláma M.: *Using Neural Nets to Learn Weights of Rules for Compositional Expert Systems.* IEA-98-AIE, 1998, s. 511–519.

Berka P.: Smart Adaptive Features of Machine Learning Methods, *EUNITE Technical Report*, 2001.

Berka P.: *Dobývání znalostí z databází*. Academia, Praha 2003.

Clark P., Niblett T.: The CN2 induction algorithm, *Machine Learning*, 3, 1989, s. 261–283.

Cortes C., Vapnik V.: Support vector networks, *Machine Learning*, Vol. 20, 1995, s. 273–297.

Dietterich T. G.: An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization, *Machine Learning*, 40(2), 2000, s. 139–158.

Duda R. O., Gasching J. E.: Model design in the Prospector consultant system for mineral exploration. In: Michie D., ed.: *Expert Systems in the Micro Electronic Age*. Edinburgh University Press, 1979.

Fayyad U., Piatetsky-Shapiro G., Smyth P., Uthurusamy R., eds: *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press, 1996.

Hand D., Manilla H., Smyth P.: *Principles of Data Mining*. MIT Press, 2002.

Harries M., Sammut C., Horn K.: Extracting Hidden Context, *Machine Learning*, 32, 1998, s. 101–126.

Kotek Z., Chalupa V., Brůha I., Jelínek J.: *Adaptivní a učící se systémy*. SNTL, Praha 1980.

Mahoney J. J., Mooney R. J.: Comparing Methods for Refining Ceretainty-Factor Rule-Bases. In: *Proc. ML94*, 1994.

Maloof M., Michalski R.: Incremental Learning with Partial Instance Memory, *Artificial Intelligence* 154, 2004, s. 95–126.

Matwin S., Kubat M.: The Role of Context in Concept Learning. In: Kubat, Widmer, eds: *Workshop on Learning in Context-Sensitive Domains*. Int. Conf. on Machine Learning ICML'96, 1996.

Michalski R., Carbonell J., Mitchell T.: *Machine Learning: An Artificial Intelligence Approach*. Tioga Publ., 1983.

Mitchell T.: *Machine Learning*. McGraw-Hill, 1997.

Muggleton S., ed.: *Inductive Logic Programming*. Academic Press, 1992.

Nunez M., Fidalgo R., Morales R.: Learning in Environments with Unknown Dynamics: Towards more Robust Concept Learners, *Journal of Machine Learning Research*, 8, 2007, s. 2595–2628.

Ourston D., Mooney R.: Theory Refinement Combining Analytical and Empirical Methods, *Artificial Intelligence*, 66, 1994, s. 311–344.

Quinlan J. R.: Induction of decision trees, *Machine Learning*, 1(1), 1986, s. 81–106.

Ruping S.: Incremental Learning with Support Vector Machines. In: *Proc. ICDM 2001*, 2001.

Schlimmer J., Fischer D.: A Case Study of Incremental Concept Induction. In: *Proc. IJCAI'86*, 1986.

Schlimmer J. C., Granger R. H.: Incremental Learning from Noisy Data, *Machine Learning*, 1, 1986, s. 317–354.

Torgo L.: YAILS: an Incremental Learning Program, *LIACC Tech. Report*, 92.1., 1992.

Towell G. G., Shavlik J.: Extracting refined rules from knowledge based neural networks, *Machine Learning*, 13(1), 1993, s. 71–102.

Tsumoto S., Tanaka H.: Incremental Learning Probabilistic Rules from Clinical Databases based on Rough Set Theory, *Journal of AMIA*, 4, 1997, s. 198–202.

Tsymbal A.: *The problem of concept drift: definitions and related work*. Technical Report TCD-CS-2004-15, Department of Computer Science, Trinity College Dublin, 2004.

Turney P. D.: The Identification of Context-sensitive Features: a Formal Definition of Context for Concept Learning. In: Kubat, Widmer, eds: *Workshop on Learning in Context-Sensitive Domains.* Int. Conf. on Machine Learning ICML'96, 1996.

Utgoff P. E., Berkman N. C., Clouse J. A.: Decision tree induction based on efficient tree restructuring, *Machine Learning*, 29, 1997, s. 5–44.

Watson I., Marir F.: Case-Based Reasoning: A Review. *Knowledge Engineering Review*, 9, 1994, s. 327–354.

Widmer G., Kubat M.: Learning in the Presence of Concept Drift and Hidden Contexts, *Machine Learning*, 23(1), 1996, s. 69–101.

Witten I. H., Frank E.: *Data Mining. Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufman, 1999.

11 Metoda GUHA a dobývání znalostí z databází

11.1 Úvod

GUHA je původní česká metoda explorační analýzy dat. Její princip spočívá v *nabízení všech vztahů, které jsou platné v daných datech a relevantní k danému problému.* Metoda je realizována pomocí řady programů nazývaných GUHA procedura. Vstupem GUHA procedury jsou analyzovaná data a několik parametrů definujících rozsáhlou množinu relevantních vztahů, výstupem procedury jsou všechna prostá tvrzení. Prosté tvrzení je takový relevantní vztah, který je pravdivý v analyzovaných datech a neplyne z jiného, jednoduššího a pravdivého vztahu již ve výstupu uvedeného. Vývoj a aplikace metody započaly v šedesátých letech minulého století (Hájek a kol., 1966). Byly vytvořeny ucelené matematické základy automatické tvorby hypotéz využívající matematickou logiku a statistiku (Hájek a Havránek, 1978). Byla implementována řada GUHA procedur, které byly mnohokrát aplikovány. Historie metody GUHA je popsána v práci (Hájek, 2004), kde je i bibliografie do roku 2003.

Dobývání znalostí z databází (dále pouze DZD) zde chápeme jako *netriviální získávání implicitních, dříve neznámých a potenciálně užitečných informací z dat* (Fayyad a kol., 1996) a (Berka, 2003). Prudký rozvoj DZD jako disciplíny informatiky započal v devadesátých letech minulého století, kdy se stal potenciál skrytý v rozsáhlých, někdy i desítky let provozovaných databázích, zřejmý jejich majitelům. Základem DZD je propojení výsledků ze tří oblastí – databází, statistiky a strojového učení.

Již ze stručné charakteristiky metody GUHA a DZD je zřejmé, že mezi nimi existují styčné body. Z dnešního pohledu je GUHA metodou DZD, je však třeba zdůraznit, že GUHA je starší než DZD a má ucelené teoretické základy. Cílem této kapitoly je jednak poukázat na důležité styčné body a jednak naznačit možnosti využití výsledků dosažených při vývoji metody GUHA v DZD. Cílem není podat úplnou historii metody GUHA, její vývoj do roku 2003 podrobně zpracován v práci (Hájek, 2004). Je zde však poukázáno na některé důležité výsledky dosažené po roce 2003 a na jejich vztah k DZD. Významu metody GUHA pro DZD se věnuje také článek (Hájek a kol., 2010), kde je i další rozsáhlá bibliografie.

Spojitost metody GUHA s DZD je nejvíc patrná v souvislosti s asociačními pravidly. Asociační pravidlo je jedna z nejznámějších forem znalostí zmiňovaných v souvislosti s DZD. Asociační pravidla byla zpopularizována počátkem devadesátých let minulého století jako prostředek pro analýzu dat o nákupních košících (Agrawal a kol., 1993), (Agrawal a kol., 1996). Algoritmus *apriori*, který se široce používá pro získávání asociačních pravidel, je podle Xindonga (Xindong a kol., 2007) čtvrtým nejvíce používaným

algoritmem v DZD. Nejvíce používanou GUHA procedurou je procedura ASSOC (Hájek a kol., 1978). Vztahy, které procedura ASSOC poskytuje již od sedmdesátých let minulého století, chápeme také jako asociační pravidla, byť jsou podstatně obecnější než asociační pravidla definovaná v devadesátých letech v sborníku (Agrawal a kol., 1993). Poměrně široce používaná implementace procedury ASSOC pro sálové počítače byla zveřejněna již v roce 1976 (Hájek a kol., 1976), (Hájek a kol., 1983). V současné době je asi nejvíce používanou implementací procedury ASSOC procedura 4ft-Miner (Rauch, Šimůnek, 2005a), existují ale i jiné implementace (Hájek, 2004), (Ralbovský, Kuchař, 2007). Vztah GUHA procedury zvané ASSOC k asociačním pravidlům definovaným v souvislosti s analýzou nákupního košíku je podrobněji popsán v oddílu 11.2.

Procedura ASSOC poskytuje obecnější asociační pravidla, než jsou asociační pravidla uvedená v knize (Agrawal a kol., 1996). Dalším rozdílem je, že všechny zmíněné implementace procedury ASSOC vycházejí z reprezentace analyzovaných dat pomocí vhodných bitových řetízků (Rauch, 1978), nepoužívá se algoritmus apriori běžně využívaný při analýze dat o nákupních košících. Tento přístup umožňuje velmi jemně definovat množinu relevantních asociačních pravidel. Rozsáhlé možnosti definice množiny relevantních asociačních pravidel jsou zahrnuty v proceduře 4ft-Miner, která je součástí systému LISP-Miner (Šimůnek, 2003). Softwarové moduly pro práci s bitovými řetízky umožnily implementaci šesti dalších GUHA procedur (Rauch, Šimůnek, 2005b), které jsou také součástí systému. Systém LISp-Miner je popsán v oddílu 11.3.

Významným rysem metody GUHA je její teoretický základ vycházející z matematické logiky a statistiky. Důležité výsledky se týkají observačních kalkulů, tedy logických kalkulů, jejichž formule odpovídají tvrzením o analyzovaných datech. Mezi takové výsledky patří například dedukční pravidla týkající se logického vyplývání mezi tvrzeními o analyzovaných datech. Výsledky, dosažené v šedesátých a sedmdesátých letech, jsou shrnuty v monografii (Hájek, 1978). Důležité výsledky byly získány i v posledních letech, např. (Rauch, 2005), (Rauch, 2010), (Rauch, 2013). Hlavní rysy observačních kalkulů jsou shrnuty v oddílu 11.4.

Metoda GUHA je používána k výuce na několika vysokých školách i při řešení reálných problémů. Aplikace často inspirují k výzkumu v různých dílčích směrech, které jsou v souladu se směry výzkumu, na jejichž důležitosti se shodují odborníci v DZD (Qiang, Xindong, 2006). Informace o současném výzkumu spojeném s metodou GUHA jsou shrnuty v oddílu 11.5, viz též (Rauch, 2013).

11.2 GUHA a asociační pravidla

Nejvíce rozšířenou a zároveň jednou z nejstarších GUHA procedur je procedura ASSOC. Jejím vstupem je matice dat spolu se zadáním množiny asociačních pravidel, která mohou být zajímavá k danému problému. Výstupem jsou všechna asociační pravidla ze zadané množiny, která jsou pravdivá v zadané matici. Základní rysy procedury ASSOC jsou uvedeny v odst. 11.2.1. Následně pak v odst. 11.2.2 najdeme příklady aplikací procedury 4ft-Miner, která je jednou z implementací procedury ASSOC. V praxi se běžně stává, že v analyzovaných datech je neúplná informace, tedy že některé údaje chybí. V takovém případě je v proceduře ASSOC možné zvolit některý ze tří přístupů – vyne-

chání chybějící informace, zabezpečené doplnění nebo optimistické doplnění, viz odst. 11.2.3.

Jak již bylo uvedeno, vztahy, které procedura ASSOC poskytuje, jsou podstatným zobecněním asociačních pravidel definovaných a rozsáhle zkoumaných v souvislosti s analýzou dat o nákupních košících (Agrawal a kol., 1993), (Agrawal a kol., 1996). Pojem asociační pravidlo se pro vztahy poskytované procedurou ASSOC začal používat vzhledem k jeho popularitě. Základní informace o vztahu procedury ASSOC a asociačních pravidel pro analýzu nákupních košíků jsou uvedeny v odst. 11.2.4.

11.2.1 Procedura ASSOC

Procedura ASSOC pracuje s maticemi dat. Řádky matice odpovídají sledovaným objektům, sloupce atributům charakterizujícím jednotlivé objekty. Příkladem je matice M, viz obr. 11.1.

pacient	Atributy				Příklady booleovských atributů vytvořených z atributů – sloupců matice			
					Základní			Odvozené
	Město	Věk	Výška	...	Město(Plzeň)	Město(Aš, Bor)	Věk⟨21, 30⟩	Město(Plzeň) ∧ Věk⟨21, 30⟩
o_1	Plzeň	27	180	...	1	0	1	1
o_2	Aš	44	186	...	0	1	0	0
o_3	Bor	31	171	...	0	1	0	0
...	...	...	...	...	...	...	...	...
o_{950}	Praha	30	177	...	0	0	1	0

Obr. 11.1 Matice dat M spolu s příklady booleovských atributů.

Řádky matice M z obr. 11.1 odpovídají pacientům, sloupce atributům popisujícím pacienty. V obr. 11.1 jsou uvedeny tři atributy – Město, Věk a Výška, ostatní jsou pouze naznačeny. První pacient o_1 je tedy z Plzně, jeho věk je 27 let a měří 180 centimetrů atd.

V obr. 11.1 jsou uvedeny i příklady booleovských atributů odvozených ze sloupců matice. Jsou zde tři *základní booleovské atributy* – Město(Plzeň), který říká, že pacient je z Plzně, Město(Aš, Bor) který říká, že pacient je z Aše nebo z Boru a Věk⟨21, 30⟩, který říká, že věk pacienta je z intervalu ⟨21, 30⟩. Základní booleovské atributy Město(Plzeň) a Město(Aš, Bor) jsou vytvořeny z atributu Město. Základní booleovský atribut Věk⟨21, 30⟩ je vytvořen z atributu Věk. Dále je v obr. 11.1 uveden *odvozený booleovský atribut* Město(Plzeň) ∧ Věk⟨21, 30⟩, který je konjunkcí základních booleovských atributů Město(Plzeň) a Věk⟨21, 30⟩. Je-li booleovský atribut pro pacienta pravdivý, je v příslušném řádku a sloupci uvedeno „1", je-li nepravdivý, je v něm zaznamenána „0".

Procedura ASSOC pracuje se základními a odvozenými booleovskými atributy. Každý základní booleovský atribut je vytvořen z jednoho atributu – sloupce analyzované matice M. Předpokládejme, že atribut A má přípustné hodnoty $a_1, a_2, \ldots, a_k$. Přípustné hodnoty atributu se nazývají *kategorie*. Každý základní booleovský atribut vytvořený z A má tvar $A(\alpha)$, kde $\alpha \subset \{a_1, \ldots, a_k\}$ je vlastní podmnožina množiny všech kategorií. Podmnožina α se nazývá *koeficient* základního booleovského atributu $A(\alpha)$.

Tedy $A(a_1)$ a $A(a_2, a_3)$ jsou příklady základních booleovských atributů vytvořených z atributu A. Základní booleovský atribut $A(\alpha)$ *je pravdivý pro řádek* **o** *matice* M, pokud $A(o) \in \alpha$, tedy pokud hodnota $A(o)$ atributu A v řádku **o** patří do množiny α. V opačném případě, pokud $A(o) \notin \alpha$, je základní booleovský atribut $A(\alpha)$ nepravdivý pro řádek **o** matice M. Například základní booleovský atribut Město(Aš, Bor) je pravdivý pro řádky o_2 a o_3 matice M uvedené na obr. 11.1 a nepravdivý pro řádky o_1 a o_{950} téže matice. Poznamenejme, že podle právě uvedené definice bychom základní booleovské atributy $A(a_1)$ a $A(a_2, a_3)$ měli psát ve formě $A(\{a_1\})$ a $A(\{a_2, a_3\})$, budeme však používat jednodušší tvar $A(a_1)$ a $A(a_2, a_3)$. Podobně budeme psát Věk(21, 30⟩ místo Věk($\{\langle 21, 30\rangle\}$) atd.

Odvozené booleovské atributy se vytvářejí ze základních booleovských atributů pomocí booleovských spojek $\neg$, $\wedge$ a $\vee$. Příklady odvozených booleovských atributů jsou $\neg A(a_1)$, $A(a_2, a_3) \wedge B(b_1)$, $B(b_3) \vee C(c_1)$ a $A(a_4) \wedge (B(b_3) \vee C(c_1))$. Pravdivost odvozených booleovských atributů se určuje obvyklým způsobem v závislosti na použitých booleovských spojkách. V dalším textu budeme *booleovským atributem* chápat základní nebo odvozený booleovský atribut. Pokud nebude nebezpečí nedorozumění, budeme místo booleovský atribut používat pouze výraz atribut.

Asociační pravidla, se kterými pracuje procedura ASSOC, mají tvar $\varphi \approx \psi$. Zde φ a ψ jsou booleovské atributy vytvořené ze sloupců analyzované matice dat matice M, φ se nazývá *antecedent* a ψ pak *sukcedent*. Symbol $\approx$ je *4ft-kvantifikátor*, jde o speciální případ zobecněného kvantifikátoru zavedeného v článku (Hájek, Havránek, 1978). 4ft-kvantifikátor $\approx$ definuje vztah mezi antecedentem a sukcedentem, který asociační pravidlo $\varphi \approx \psi$ vyjadřuje. Existuje mnoho různých 4ft-kvantifikátorů, každému 4ft-kvantifikátoru je přiřazena podmínka týkající se kontingenčních tabulek dvou booleovských atributů.

Asociační pravidlo $\varphi \approx \psi$ je *pravdivé v matici* M, pokud podmínka přiřazená 4ft-kvantifikátoru $\approx$ je splněna pro kontingenční tabulku atributů φ a ψ v matici M. V opačném případě je asociační pravidlo $\varphi \approx \psi$ *nepravdivé v matici* M. Je-li $\varphi \approx \psi$ pravdivé v matici M, píšeme $\mathrm{Val}(\varphi \approx \psi, M) = 1$, je-li nepravdivé píšeme $\mathrm{Val}(\varphi \approx \psi, M) = 0$. Kontingenční tabulku atributů φ a ψ v matici M značíme $4ft(\varphi, \psi, M)$. Jde o čtveřici čísel $\langle a, b, c, d\rangle$, viz tab. 11.1.

Tabulka 11.1 Kontingenční tabulka $4ft(\varphi, \psi, M)$ atributů φ a ψ v matici dat M

M	ψ	$\neg\psi$
φ	a	b
$\neg\varphi$	c	d

Zde a je počet řádků matice M, pro které jsou pravdivé φ i ψ, b udává počet řádků, pro které je φ pravdivé a ψ nepravdivé, c značí počet řádků, pro které je φ nepravdivé a ψ pravdivé, a d představuje počet řádků, pro které jsou φ i ψ nepravdivé. Kontingenční tabulku $4ft(\varphi, \psi, M)$ atributů φ a ψ v matici M také někdy nazýváme čtyřpolní tabulkou, píšeme také $4ft(\varphi, \psi, M) = \langle a, b, c, d\rangle$.

Nejčastěji používané 4ft-kvantifikátory jsou definovány v pracích (Hájek, Havránek, 1978), (Hájek a kol., 1983), (Rauch, Šimůnek, 2005a), většina z nich je implementována

v proceduře 4ft-Miner. Krom toho lze definovat další zajímavé 4ft-kvantifikátory na základě různých měr zajímavosti asociačních pravidel (Rauch, 2013). Dále jsou uvedeny definice čtyř často používaných 4ft-kvantifikátorů, některé z nich jsou použity v odst. 11.2.2.

4ft-kvantifikátor $\Rightarrow_{p;Base}$ *fundované implikace* je pro $0 < p \leq 1$ a $Base > 0$ definován, viz (Hájek, Havránek, 1978), podmínkou $\frac{a}{a+b} \geq p \wedge a \geq Base$. To znamená, že asociační pravidlo $\varphi \Rightarrow_{p;Base} \psi$ je pravdivé v matici dat M, pouze pokud alespoň $100p$ procent z řádků splňujících booleovský atribut φ splňuje i booleovský atribut ψ a pokud zároveň nejméně *Base* řádků splňuje jak φ, tak i ψ. Poznamenejme, že podíl $\frac{a}{a+b}$ se nazývá konfidence, viz též odst. 11.2.4.

4ft-kvantifikátor $\Leftrightarrow_{p;Base}$ *dvojité fundované implikace* je pro $0 < p \leq 1$ a $Base > 0$ definován, viz (Hájek a kol., 1983), podmínkou $\frac{a}{a+b+c} \geq p \wedge a \geq Base$. To znamená, že asociační pravidlo $\varphi \Leftrightarrow_{p;Base} \psi$ je pravdivé v matici dat M, pouze pokud alespoň $100p$ procent z řádků splňujících φ nebo ψ splňuje jak φ, tak i ψ a zároveň nejméně *Base* řádků splňuje φ i ψ.

4ft-kvantifikátor $\equiv_{p;Base}$ *fundované ekvivalence* je pro $0 < p \leq 1$ a $Base > 0$ definován, viz (Hájek a kol., 1983), podmínkou $\frac{a+d}{a+b+c+d} \geq p \wedge a \geq Base$. To znamená, že asociační pravidlo $\varphi \equiv_{p;Base} \psi$ je pravdivé v matici dat M, pouze pokud alespoň $100p$ procent z řádků má stejnou hodnotu (buď *pravda*, nebo *nepravda*) pro oba booleovské atributy φ nebo ψ a zároveň nejméně *Base* řádků splňuje φ i ψ.

4ft-kvantifikátor $\Rightarrow^{+}_{p;Base}$ *nadprůměrné závislosti*, který se často nazývá *AA-kvantifikátor* (Above Average Dependence) je pro $0 < p$ a $Base > 0$ definován, viz (Rauch, Šimůnek, 2005a), podmínkou $\frac{a}{a+b} \geq (1+p)\frac{a+c}{a+b+c+d} \wedge \; a \geq Base$. To znamená, že asociační pravidlo $\varphi \Rightarrow^{+}_{p;Base} \psi$ je pravdivé v matici dat M, pouze pokud relativní četnost řádků splňujících booleovský atribut ψ mezi řádky splňujícími booleovský atribut φ je alespoň o $100p$ procent větší než relativní četnost řádků splňujících ψ v celé matici a pokud zároveň nejméně *Base* řádků splňuje jak φ, tak i ψ. Vynecháme-li podmínku $a \geq Base$, pak lze říci, že AA-kvantifikátor vyjadřuje nárůst relativní četnosti alespoň o $100p$ procent, pokud je splněna podmínka daná booleovským atributem φ.

Pokud je pro podmínka přiřazená 4ft-kvantifikátoru $\approx$ splněna ve čtyřpolní tabulce $\langle a, b, c, d \rangle$, tak píšeme $\approx (a, b, c, d) = 1$, není-li splněna, píšeme $\approx (a, b, c, d) = 0$.

11.2.2 Příklady aplikací procedury 4ft-Miner

Pro demonstraci možností procedury ASSOC použijeme proceduru 4ft-Miner, která je nejvíce používanou současnou implementací procedury ASSOC a zároveň má bohatší možnosti definice množiny relevantních asociačních pravidel než dřívější implementace. Příklady jsou zvoleny tak aby, jednak ukazovaly podstatné rysy procedury 4ft-Miner a jednak naznačily možnosti využití metody GUHA v celém procesu DZD.

Dobývání znalostí z databází je složitým procesem skládajícím se z více vzájemně provázaných kroků s mnoha zpětnými vazbami. Aplikace analytické procedury, v našem případě procedury 4ft-Miner, je pouze jeden z těchto kroků. Pro řízení procesu je k dispozici několik metodik, velmi známou je metodologie CRISP-DM (Berka, 2003). Poznamenejme, že za jednu z prvních metodik DZD lze považovat popis procesu aplikace GUHA procedur uvedený v knize (Hájek a kol., 1983).

Podle metodologie CRISP-DM se proces DZD dělí na šest hlavních etap. Prvními dvěma jsou *porozumění problematice* a *porozumění datům*. Mezi těmito dvěma etapami je, stejně jako mezi všemi dalšími, řada zpětných vazeb. Je třeba začít uvedenými dvěma etapami, výsledkem je *analytická otázka*, kterou je třeba vyřešit. Na základě analytické otázky je třeba vybrat analytickou proceduru a v etapě *příprava dat* provést potřebné transformace dat a na výsledná data v etapě *modelování* použít vybranou analytickou proceduru. Následují etapy *interpretace* a *využití výsledků*, popř. v rámci zpětné vazby návrat k některé z předchozích etap. V našich příkladech si budeme všímat analytických otázek formulovaných s ohledem na možnosti GUHA procedury nazývané 4ft-Miner.

Příklady se týkají medicínské problematiky, konkrétně studia rizikových faktorů ateroskleróza u mužů středního věku. Využijeme k tomu data získaná v projektu STULONG (Berka a kol., 2009) a (Rauch, Tomečková, 2007). Jeho podstatou je dlouhodobé sledování ateroskleróza u mužů středního věku. Použijeme výsledky vstupního vyšetření 1417 mužů. Byly zjišťovány hodnoty 244 atributů pro každého pacienta, 64 atributy jsou číselnými výsledky měření nebo kódy ať už získanými přímo při vyšetření, nebo jako výsledky transformací ostatních atributů. Hodnoty těchto 64 atributů pro 1417 mužů tvoří matici dat *Vstup*, která má 1417 řádků a 64 sloupců.[1]

Pro účely prezentace možností procedury 4ft-Miner vycházíme z předpokladu, že matice dat *Vstup* dosud nebyla analyzována. Přirozenou analytickou otázkou tedy je *„Jaké zajímavé vztahy platí v matici dat Vstup?"*. Připomeňme, že princip metody GUHA spočívá v nabízení všech vztahů, které jsou jednak platné v daných datech a jednak relevantní k danému problému. Vstupem GUHA procedury jsou analyzovaná data a parametry definující rozsáhlou množinu relevantních vztahů. Výstupem jsou všechna prostá tvrzení, tedy tvrzení, která jsou pravdivá v analyzovaných datech a která neplynou z jiných a jednodušších tvrzení již ve výstupu uvedených. Princip metody GUHA tedy navozuje dojem, že stačí vhodně nadefinovat vstupní parametry GUHA procedury a tato přirozená analytická otázka bude zodpovězena výstupem GUHA procedury. Otázka *„Jaké zajímavé vztahy platí v matici dat Vstup?"* je však příliš obecná a naznačený přístup nevede k cíli, ale obvykle k zahlcení výsledky. Navíc všechna prostá tvrzení zahrnují i tvrzení banální nebo tvrzení všeobecně známá, která je třeba odfiltrovat, viz též výzkumná témata v oddílu 11.5.

Příliš obecnou otázku *„Jaké zajímavé vztahy platí v matici dat Vstup?"* je vhodné zúžit na dílčí otázky týkající se vždy pouze několika málo zajímavých skupin atributů. Skupiny atributů jsou jedním z prostředků, jak do analýz zahrnout prvky sémantiky. Skupiny jsou obvykle definovány odborníky z příslušné oblasti, takto definované skupi-

[1] Projekt STULONG byl realizován na II. interní klinice 1. lékařské fakulty Univerzity Karlovy a ve Všeobecné fakultní nemocnici v Praze pod vedením prof. MUDr. F. Boudíka, DrSc., ve spolupráci s MUDr. M. Tomečkovou, CSc., a prof. MUDr. J. Bultasem, CSc. Data byla převedena do elektronické podoby Evropským centrem pro medicínskou informatiku, statistiku a epidemiologii Univerzity Karlovy a Akademie věd ČR pod vedením prof. RNDr. Jany Zvárové, DrSc.

ny odrážejí obecně přijímané členění analyzované oblasti a k těmto skupinám se váží expertní znalosti.

Příkladem skupin atributů jsou skupiny definované v datech STULONG, viz (Berka a kol., 2009). Pro účely dále uvedených příkladů však použijeme skupiny atributů, které vznikly modifikací těchto skupin. Cílem je demonstrovat důležité rysy procedury 4ft-Miner, ale ne nalezení nových medicínsky zajímavých znalostí. Použijeme čtyři skupiny atributů – ***Osobní charakteristiky***, ***Tělesné aktivity***, ***Měření*** a ***Rizika***. Přehled atributů všech čtyř skupin je zaznamenán v tab. 11.2. Pro každý atribut je uvedeno jeho jméno, krátké jméno a přípustné kategorie spolu s jejich frekvencemi, je uváděn i počet pacientů s chybějící informací. Jméno atributu pro naše účely atribut dostatečně popisuje, krátké jméno je uváděno místo jména v příkladech zadání a výsledků. Je třeba se zmínit, že data STULONG se týkají pouze mužů středního věku, proto se neudává pohlaví a atribut Věk má pouze čtyři kategorie.

Tabulka 11.2 Přehled atributů skupin Osobní charakteristiky, Tělesné aktivity, Měření a Rizika

Skupina	Atribut		*Kategorie*
	Jméno	Krátké jméno	název – počet pacientů / název – počet pacientů / ...
Osobní charakteristiky	Stav	Stav	ženatý – 1207 / rozvedený – 104 / svobodný – 95 / vdovec – 10 / chybí – 1
	Věk	Věk	⟨35; 40⟩ – 111 / ⟨40; 45⟩ – 450 / ⟨45; 50⟩ – 680 / ⟨50; 55⟩ – 176
	Vzdělání	Vzdělání	základní – 151 / vyučen – 405 / středoškolské – 444 / vysokoškolské – 397 / chybí – 20
	Zodpovědnost v zaměstnání	Zodpovědnost	řídicí pracovník – 286 / částečně samostatný pracovník – 435 / ostatní – 636 / důchodce pro ICHS – 6 / důchodce, jiné důvody – 19 / chybí – 35
Tělesné aktivity	Tělesná aktivita v zaměstnání	Telaktza	převážně sedí – 739 / převážně stojí – 167 / převážně chodí – 373 / přenáší těžká břemena – 100 / chybí – 38
	Tělesná aktivita po zaměstnání	Aktpozam	převážně sedí – 266 / mírná – 1028 / velká – 118 / chybí – 5
Měření	Body Mass Index	BMI	13 kategorií, viz tab. 11.3 a obr. 11.2
	Cholesterol v g%	Cholesterol	19 kategorií, viz tab. 11.4 a obr. 11.3
Rizika	Diabetes	Diabetes	ano – 30 / ne – 1378 / chybí – 9
	Hypertenze	Hypertenze	ano – 220 / ne – 1192 / chybí – 5
	Hyperlipidemie	Hyperlipidemie	ano – 54 / ne – 815 / chybí – 548
	Infarkt myokardu	Infarkt	ano – 34 / ne – 1378 / chybí – 5

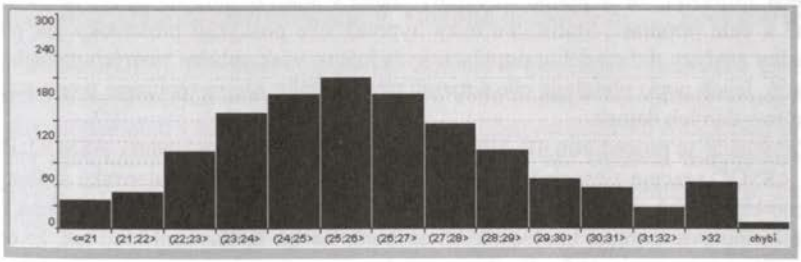
V tabulce 11.2 jsou uvedeny i frekvence kategorií pro většinu atributů, pro atributy BMI a Cholesterol jsou zaznamenány frekvence v tabulkách 11.3 a 11.4. Na obrázcích 11.3 a 11.4 jsou uvedeny histogramy těchto frekvencí, jde zároveň o ukázku výstupů modulu LMDataSource, který je součástí systému LISp-Miner a slouží pro vstupní transformace dat. Poznamenejme, že data STULONG jsou data s neúplnou informací.

Tabulka 11.3 Frekvence kategorií atributu Body Mass Index

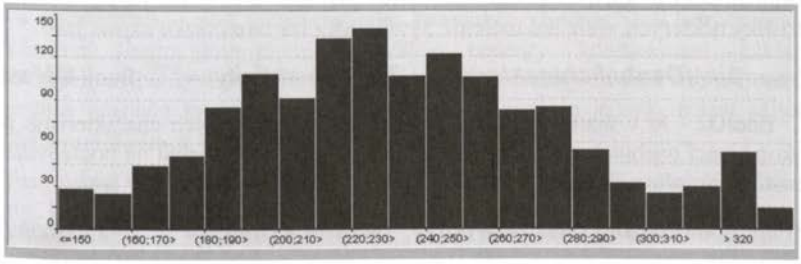
interval	≤ 21	(21; 22⟩	(22; 23⟩	(23; 24⟩	(24; 25⟩	(25; 26⟩	(26; 27⟩
pacientů	39	50	106	161	187	210	187
interval	(27; 28⟩	(28; 29⟩	(29; 30⟩	(30; 31⟩	(31; 32⟩	> 32	chybí
pacientů	145	108	68	56	28	64	8

Tabulka 11.4 Frekvence kategorií atributu Cholesterol v mg%

interval	≤ 150	(150; 160⟩	(160; 170⟩	(170; 180⟩	(180; 190⟩
pacientů	28	25	44	51	85
interval	(190; 200⟩	(200; 210⟩	(210; 220⟩	(220; 230⟩	(230; 240⟩
pacientů	108	91	133	140	107
interval	(240; 250⟩	(250; 260⟩	(260; 270⟩	(270; 280⟩	(280; 290⟩
pacientů	122	106	83	85	55
interval	(290; 300⟩	(300; 310⟩	(310; 320⟩	> 320	chybí
pacientů	32	25	29	53	15



Obr. 11.2. Frekvence kategorií atributu Body Mass Index.



Obr. 11.3. Frekvence kategorií atributu Cholesterol v mg%.

Na základě skupin z tabulky 11.2 je v odstavci 11.2.2.1 formulováno několik analytických otázek řešitelných pomocí procedury 4ft-Miner. Asociační pravidlo zachycuje vztah dvou booleovských atributů odvozených z atributů – sloupců – analyzované matice dat. Odvozené booleovské atributy vhodně charakterizují zvolené skupiny atributů. Způsob odvození booleovských charakteristik jednotlivých skupin atributů je součástí zadání procedury 4ft-Miner. Booleovské charakteristiky jednotlivých skupin použité v příkladech jsou popsány v odst. 11.2.2.2. Jednotlivým analytickým otázkám a jejich řešení pomocí procedury 4ft-Miner jsou věnovány odstavce 11.2.2.3 až 11.2.2.5.

11.2.2.1 Analytické otázky

Možnosti procedury 4ft-Miner naznačíme řešením analytických otázek týkajících se skupin atributů ***Osobní charakteristiky***, ***Tělesné aktivity***, ***Měření*** a ***Rizika***. Na jejich základě lze formulovat například analytické otázky O1 a O2:

O1 – Je v matici dat *Vstup* patrný vliv některých kombinací osobních charakteristik a tělesných aktivit pacientů na některé výsledky měření? Symbolicky lze tuto otázku zapsat jako

$$Vstup\text{: } (\textbf{\textit{Osobní charakteristiky}}, \textbf{\textit{Tělesné aktivity}}) \approx^{?} \textbf{\textit{Měření}}.$$

O2 – Je v matici dat *Vstup* patrný vliv některých kombinací osobních charakteristik a tělesných aktivit pacientů na výskyt některých rizik? Symbolicky lze tuto otázku zapsat jako

$$Vstup\text{: } (\textbf{\textit{Osobní charakteristiky}}, \textbf{\textit{Tělesné aktivity}}) \approx^{?} \textbf{\textit{Rizika}}.$$

Je třeba zdůraznit, že se ptáme na vztahy platné v analyzovaných datech, ne na vztahy platné v celé populaci. Statistické testy hypotéz sice poskytují prostředky jak přenést výsledky analýzy dat na celou populaci, vyžaduje to však splnění různých dalších předpokladů. Jejich popis přesahuje cíle a rozsah této kapitoly, zde se věnujeme pouze vztahům platným v daných datech.

Zabýváme se procedurou 4ft-Miner jakožto implementací procedury ASSOC. Procedura ASSOC pracuje s asociačními pravidly $\varphi \approx \psi$, φ a ψ jsou booleovské atributy vytvořené ze sloupců analyzované matice dat M. To znamená, že obě otázky je třeba vhodně přeformulovat pomocí booleovských atributů. To lze udělat například tak, že otázku O1 přeformulujeme na otázku BoolO1, podobnou úpravu provedeme pro O2:

BoolO1 – Je v matici dat *Vstup* patrný vliv booleovských charakteristik některých kombinací osobních charakteristik a tělesných aktivit pacientů na booleovské charakteristiky některých výsledků měření? Symbolicky lze tuto otázku zapsat jako

$$Vstup\text{: Bool}(\textbf{\textit{Osobní charakteristiky}}, \textbf{\textit{Tělesné aktivity}}) \approx^{\text{Bool?}} \text{Bool}(\textbf{\textit{Měření}}).$$

BoolO2 – Je v matici dat *Vstup* patrný vliv booleovských charakteristik některých kombinací osobních charakteristik a tělesných aktivit pacientů na booleovské charakteristiky výskytu některých rizik? Symbolicky lze tuto otázku zapsat jako

$$Vstup\text{: Bool}(\textbf{\textit{Osobní charakteristiky}}, \textbf{\textit{Tělesné aktivity}}) \approx^{\text{Bool?}} \text{Bool}(\textbf{\textit{Rizika}}).$$

Smyslem symbolických zápisů je jednak zestručnit vyjadřování a jednak příprava k budoucí formalizaci a automatizaci aplikací procedury 4ft-Miner. Procedura ASSOC pracuje s asociačními pravidly ve tvaru $\varphi \approx \psi$, booleovský atribut φ se nazývá antecedent, booleovský atribut ψ pak sukcedent, symbol $\approx$ značí 4ft-kvantifikátor. Používá se i název *cedent*, který může znamenat antecedent i sukcedent. Procedura 4ft-Miner pracuje i s *dílčími cedenty*, které lze s výhodou použít pro jemnější zadání parametrů.

Možnosti použití dílčích cedentů naznačíme dále na příkladech. Každá ze čtyř skupin atributů bude odpovídat jednomu dílčímu cedentu. To nám umožní velmi jemně definovat pro každou skupinu atributů její booleovské charakteristiky. Pro skupinu atributů ***Osobní charakteristiky*** definujeme množinu $\mathcal{B}$(***Osobní charakteristiky***) téměř

1500 booleovských atributů tak, že každý z nich bude vhodným způsobem charakterizovat tuto skupinu atributů. Podobně definujeme skupinu atributů $\mathscr{B}$(**Tělesná aktivita**), $\mathscr{B}$(**Měření**) a $\mathscr{B}$(**Rizika**).

Analytickou otázku Bool(O1) budeme řešit tak, že budeme hledat všechna asociační pravidla $\varphi \approx \psi$ pravdivá v matici dat *Vstup* taková, že $\varphi = \varphi_O \wedge \varphi_A$, kde φ_O patří do $\mathscr{B}$(**Osobní charakteristiky**), φ_A do $\mathscr{B}$(**Tělesná aktivita**), ψ do $\mathscr{B}$(**Měření**) a $\approx$ je vhodný 4ft-kvantifikátor. Použijeme postupně dva 4ft-kvantifikátory, 4ft-kvantifikátor $\Rightarrow_{p;Base}$ fundované implikace a 4ft-kvantifikátor $\Rightarrow^{+}_{p;Base}$ nadprůměrné závislosti. To znamená, že budeme řešit dvě jednoduché analytické otázky Bool(O1A) a Bool(O1B):

a) Bool(O1A). Pro které booleovské atributy φ popisující kombinace osobních charakteristik a tělesných aktivit pacientů a pro které booleovské atributy ψ popisující výsledky měření v matici dat *Vstup* platí, že alespoň $100p$ procent z pacientů splňujících φ splňuje i ψ a že zároveň nejméně *Base* pacientů splňuje jak φ, tak i ψ? Symbolicky lze tuto otázku zapsat jako

Vstup: Bool(**Osobní charakteristiky**, **Tělesné aktivity**) $\Rightarrow_{p,Base}$ Bool(**Měření**).

b) Bool(O1B). Pro které booleovské atributy φ popisující kombinace osobních charakteristik a tělesných aktivit pacientů a pro které booleovské atributy ψ popisující výsledky měření platí v matici dat *Vstup*, že relativní četnost pacientů splňujících ψ mezi pacienty splňujícími φ je alespoň o $100p$ procent větší než relativní četnost pacientů splňujících ψ v celé matici a že zároveň nejméně *Base* pacientů splňuje jak φ, tak i ψ. Symbolicky lze tuto otázku zapsat ve tvaru

Vstup: Bool (**Osobní charakteristiky**, **Tělesné aktivity**) $\Rightarrow^{+}_{p;Base}$ Bool(**Měření**).

Booleovské atributy – prvky množin $\mathscr{B}$(**Osobní charakteristiky**), $\mathscr{B}$(**Tělesná aktivita**) a $\mathscr{B}$(**Měření**) – jsou vytvářeny jako konjunkce vhodných základních booleovských atributů. Jinými slovy pracujeme s dílčími cedenty – konjunkcemi – základních booleovských atributů, viz odst. 11.2.2.2. Analytické otázky Bool(O1A) a Bool(O1B) mají jednak naznačit možnosti definice takových množin a jednak ukázat užitečnost obou 4ft-kvantifikátorů, $\Rightarrow_{p;Base}$ i $\Rightarrow^{+}_{p;Base}$.

Cílem řešení analytické otázky Bool(O2) je ukázat důležitost možnosti použít dílčí cedenty, které jsou disjunkcemi základních booleovských atributů. K tomu nám stačí použít pouze jeden ze dvou zmíněných kvantifikátorů, použijeme $\Rightarrow^{+}_{p;Base}$. Budeme řešit jednoduchou analytickou otázku Bool(O2B):

Pro které booleovské atributy φ popisující kombinace osobních charakteristik a tělesných aktivit pacientů a pro které booleovské atributy ψ popisující výsledky měření platí v matici dat *Vstup*, že relativní četnost pacientů splňujících ψ mezi pacienty splňujícími φ je alespoň o $100p$ procent větší než relativní četnost pacientů splňujících ψ v celé matici a že zároveň nejméně *Base* pacientů splňuje jak φ, tak i ψ? Symbolicky lze tuto otázku zapsat jako

Vstup: Bool(**Osobní charakteristiky**, **Tělesné aktivity**) $\Rightarrow^{+}_{p;Base}$ Bool(**Rizika**).

Jednoduchým analytickým otázkám Bool(O1A), Bool(O1B) a Bool(O2B) jsou věnovány odstavce 11.2.2.3 až 11.2.2.5.

11.2.2.2 Booleovské charakteristiky skupin atributů

Asociační pravidlo $\varphi \approx \psi$ se týká booleovských atributů φ a ψ odvozených z atributů – sloupců analyzované matice dat. Způsob odvození vychází z řešené analytické otázky. Připomeňme, že odvozené booleovské atributy se vytvářejí ze základních booleovských atributů. Základní atribut vytvořený z atributu A má tvar A(α), kde α je vlastní podmnožina množiny všech kategorií atributu A, α se nazývá koeficient A(α), viz odst. 11.2.1. Pro každý atribut A se zadává množina přípustných koeficientů, které mají být automaticky generovány. Tím je také pro každý atribut A dána množina $\mathcal{B}$(A) základních booleovských atributů, které z něho mají být odvozeny.

Procedura 4ft-Miner má několik možností, jak pro daný atribut A zadat množinu $\mathcal{B}$(A). Příklady uvedené na obr. 11.4 jsou voleny tak, aby ukázaly typické možnosti zadání množiny přípustných koeficientů. Je to množina $\mathcal{B}$(Stav) základních booleovských atributů odvozených z atributu Stav, množina $\mathcal{B}$(Věk) základních booleovských atributů odvozených z atributu Věk atd.

Obr. 11.4 Příklady zadání množiny přípustných koeficientů.

Na obrázku 11.4 jsou uvedeny nejen příklady zadání množin přípustných koeficientů, ale i příklady zadání booleovských charakteristik skupin atributů. Jde o množinu $\mathcal{B}$(***Osobní charakteristiky***) booleovských atributů odvozených ze skupiny ***Osobní charakteristiky***, analogické zadání platí pro množiny $\mathcal{B}$(***Tělesná aktivita***), $\mathcal{B}$(***Měření***) a $\mathcal{B}$(***Rizika***). Nejprve se však budeme zabývat příklady množin základních booleovských atributů odvozených z daného atributu.

Množina $\mathcal{B}$(Stav) základních booleovských atributů odvozených z atributu Stav je dána výrazem `Stav(subset), 1-1`. To znamená, že z atributu Stav se vytvoří všechny základní atributy Stav(α), kde α je podmnožina množiny všech kategorií atributu Stav a α obsahuje právě jednu kategorii (pedantsky řečeno 1–1 kategorii, tedy minimálně jednu a zároveň maximálně jednu kategorii). Takto jsou zadány čtyři booleovské atributy Stav(ženatý), Stav(rozvedený), Stav(svobodný) a Stav(vdovec), viz též tab. 11.2. Pokud by byl použit výraz `Stav(subset), 2-2`, bylo by zadáno šest booleovských atributů Stav(ženatý, rozvedený), Stav(ženatý, svobodný), Stav(ženatý, vdovec), Stav(rozvedený, svobodný), Stav(rozvedený, vdovec) a Stav(svobodný, vdovec). Výrazem `Stav(subset), 1-2` by bylo zadáno všech

deset uvedených základních booleovských atributů. Výrazy `Zodpovědnost(subset), 1-1` a `Aktpozam(subset), 1-1` definují analogickým způsobem množiny booleovských atributů $\mathscr{B}$(Zodpovědnost) a $\mathscr{B}$(Aktpozam). Obecně výraz `A(subset), u-v` definuje množinu základních booleovských atributů A(α), kde α se skládá z minimálně u a z maximálně v kategorií.

Množina $\mathscr{B}$(Věk) základních booleovských atributů odvozených z atributu Věk je dána výrazem `Věk(int), 1-2`. To znamená, že z atributu Věk se vytvoří všechny základní atributy Věk(α), kde α je interval 1–2 kategorií (tedy 1–2 po sobě jdoucí kategorie atributu Věk). Příklady takových booleovských atributů jsou Věk((35; 40⟩) a Věk((35; 40⟩, (40; 45⟩), které budeme psát stručněji Věk(35; 40⟩ a Věk(35; 45⟩. To znamená, že výrazem `Věk(int), 1-2` je definováno celkem 7 základních booleovských atributů Věk(35; 40⟩, Věk(35; 45⟩, Věk(40; 45⟩, Věk(40; 50⟩, Věk(45; 50⟩, Věk(45; 55⟩ a Věk(50; 55⟩.

Zbývající množiny $\mathscr{B}$(Vzdělání), $\mathscr{B}$(Zodpovědnost), $\mathscr{B}$(Telaktza) a $\mathscr{B}$(Aktpozam) ze skupin ***Osobní charakteristiky*** a ***Tělesné aktivity*** jsou definovány podobně.

Množina $\mathscr{B}$(BMI) základních booleovských atributů odvozených z atributu BMI je dána výrazem `BMI(int), 1-5`. To znamená, že z atributu BMI se vytvoří všechny základní atributy BMI(α), kde α je interval 1–5 kategorií (tedy 1–5 po sobě jdoucích kategorií atributu BMI). Atribut BMI má 13 kategorií, viz tabulky 11.2 a 11.3. To znamená, že je takto dáno 13 intervalů tvořených jednou kategorií, např. BMI(≤ 21), BMI(21; 22⟩ nebo BMI(> 32⟩, 12 intervalů tvořených dvěma kategoriemi, např. BMI(≤ 22), BMI(21; 23⟩ nebo BMI(> 31⟩ atd. až 9 intervalů tvořených pěti kategoriemi, např. BMI(≤ 25), BMI(21; 26⟩ nebo BMI(> 28⟩. Celkem je takto dáno 55 základních booleovských atributů BMI(α) s koeficienty – intervaly délky 1–5. Připomeňme, že píšeme BMI(≤ 22) místo BMI(≤ 21, (21; 22⟩) atd.

Množina $\mathscr{B}$(Cholesterol) základních booleovských atributů odvozených z atributu Cholesterol je dána výrazem Cholesterol(cut), 1–10. To znamená, že z atributu Cholesterol se vytvoří všechny základní atributy Cholesterol(α), kde α je levý řez nebo pravý řez délky 1–10. Levý řez délky 10 je interval délky 10 tvořený deseti nejmenšími kategoriemi, lze ho chápat jako „interval odříznutý zleva", analogicky postupujeme pro ostatní levé řezy. Takto je tedy definováno deset booleovských atributů Cholesterol(≤ 150),..., Cholesterol(≤ 240). Podobně pravý řez je „interval odříznutý zprava". Celkem je tedy definováno 20 základních booleovských intervalů, řezů délky 1–10 pro atribut Cholesterol.

Množina $\mathscr{B}$(Diabetes) základních booleovských atributů odvozených z atributu Diabetes je dána výrazem `Diabetes(ano)`. To znamená, že z atributu Diabetes se vytvoří jediný základní booleovský atribut Diabetes(ano) daný kategorií ano. Podobně pro atributy Hypertenze, Hyperlipidemie a Infarkt.

Zadání množiny $\mathscr{B}$(***Osobní charakteristiky***) booleovských charakteristik skupiny ***Osobní charakteristiky*** je v obr. 11.4 dáno výrazem `Osobní charakteristiky Conj, 1-4`. To znamená, že se automaticky vytvoří všechny konjunkce z 1–4 základních booleovských atributů z množin $\mathscr{B}$(Stav), $\mathscr{B}$(Věk), $\mathscr{B}$(Vzdělání) a $\mathscr{B}$(Zodpovědnost). Z každé z těchto množin se v jedné konjunkci může vyskytovat maximálně jeden základní booleovský atribut. Konjunkce délky 1 je samostatný základ-

ní booleovský atribut. V obrázku 11.4 je zadáno téměř 1500 booleovských atributů. Příkladem booleovské charakteristiky skupiny Osobní charakteristiky je konjunkce Stav(ženatý) ∧ Věk(40; 50⟩.

Množina $\mathscr{B}$(***Tělesná aktivita***) booleovských charakteristik skupiny ***Tělesná aktivita*** je dána výrazem `Tělesná aktivita Conj, 0-2`, viz obr. 11.4. Booleovskými charakteristikami skupiny atributů ***Tělesné aktivity*** jsou konjunkce délky 0–2 vytvořené ze základních booleovských charakteristik množin $\mathscr{B}$(Telaktza) a $\mathscr{B}$(Aktpozam). Konjunkce délky 0 je pravdivá pro každého pacienta. Takto je zadáno 32 booleovských charakteristik skupiny atributů ***Tělesné aktivity***. Minimální délka konjunkce 0 umožňuje, že se booleovské charakteristiky skupiny ***Tělesná aktivita*** neberou v úvahu.

Zadání množiny $\mathscr{B}$(Měření) booleovských charakteristik skupiny Měření je na obr. 11.4 dáno výrazem `Měření Conj, 1-2`. Takto je dáno více než 1175 konjunkcí délky 1–2 vytvořených základními booleovskými charakteristikami z množin $\mathscr{B}$(BMI) a $\mathscr{B}$(Cholesterol).

Množina $\mathscr{B}$(***Rizika***) booleovských charakteristik skupiny ***Rizika*** je dána výrazem `Rizika Disj, 1-4`. Jsou to disjunkce délky 1–4 vytvořené ze základních booleovských atributů Diabetes(ano), Hypertenze(ano), Hyperlipidemie(ano) a Infarkt(ano). Takto je definováno 15 booleovských charakteristik skupiny atributů ***Rizika***.

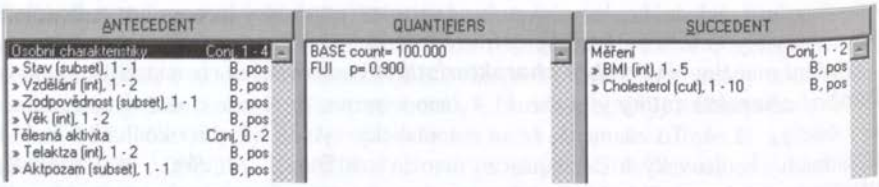
11.2.2.3 Bool(Osobní charakteristiky, Tělesné aktivity) $\Rightarrow_{p;Base}$ *Bool(Měření)*

Analytickou otázku Bool(O1A)

Vstup: Bool(***Osobní charakteristiky***, ***Tělesné aktivity***) $\Rightarrow_{p;Base}$ Bool(***Měření***)

řešíme tak, že hledáme asociační pravidla $\varphi \approx \psi$, která jsou pravdivá v matici dat *Vstup* a pro něž platí $\varphi = \varphi_O \wedge \varphi_A$, $\varphi_O \in \mathscr{B}$(***Osobní charakteristiky***), $\varphi_A \in$ (***Tělesná aktivita***) a $\psi \in \mathscr{B}$(***Měření***). 4ft-kvantifikátor $\Rightarrow_{p;Base}$ říká, že alespoň $100p$ procent z pacientů splňujících φ splňuje i ψ a že zároveň nejméně *Base* pacientů splňuje jak φ, tak i ψ. Častým počátečním požadavkem při podobných analýzách je, aby alespoň 90 % pacientů splňujících φ splňovalo i ψ. Obvyklým požadavkem také je, aby alespoň 5 % ze všech řádků analyzované matice splňovalo φ i ψ. Matice dat *Vstup* má 1417 řádků, zaokrouhlíme-li poněkud těchto 5 % nahoru, dostaneme počáteční hodnotu *Base* = 100. To znamená, že použijeme 4ft-kvantifikátor $\Rightarrow_{0,9;100}$.

Takto formulovaná analytická otázka je zadáním pro proceduru 4ft-Miner, jež se skládá z matice dat *Vstup* a ze zadání množiny relevantních asociačních pravidel, viz obr. 11.5.



Obr. 11.5 Zadání procedury 4ft-Miner pro otázku Bool(O1A) s 4ft-kvantifikátorem $\Rightarrow_{0,9;100}$

Obecně platí, že procedura 4ft-Miner generuje a v zadané matici dat verifikuje všechna asociační pravidla $\varphi \approx \psi$, kde φ patří do množiny relevantních antecedentů, jež je zadána ve sloupci ANTECEDENT, ψ patří do množiny relevantních sukcedentů, která je zadána ve sloupci SUCCEDENT, $\approx$ je 4ft-kvantifikátor zadaný ve sloupci QUANTIFIERS a φ a ψ nemají žádné společné atributy.

Množina relevantních antecedentů je určena pomocí zadání dílčích antecedentů. V obrázku 11.5 jsou ve sloupci ANTECEDENT dva dílčí antecedenty – *Osobní charakteristiky* a *Tělesná aktivita*. Jejich zadání odpovídají množinám $\mathcal{B}$(**Osobní charakteristiky**) a $\mathcal{B}$(**Tělesná aktivita**) popsaným v odst. 11.2.2.2. Každý antecedent φ je generován jako konjunkce dílčích antecedentů patřících do jednotlivých dílčích antecedentů. Dle zadání v obr. 11.5 jsou tedy generovány antecedenty – konjunkce $\varphi_O \wedge \varphi_A$, kde $\varphi_O \in \mathcal{B}$(**Osobní charakteristiky**) a $\varphi_A \in \mathcal{B}$(**Tělesná aktivita**), viz otázky Bool(O1A) v odst. 11.2.2.1.

Množina relevantních sukcedentů se zadává analogicky pomocí dílčích sukcedentů, v obr. 11.5 je použit jediný dílčí sukcedent *Měření* odpovídající množině $\mathcal{B}$(**Měření**). Ve sloupci QUANTIFIERS je zadán 4ft-kvantifikátor $\Rightarrow_{0,9;100}$.

Připomeňme, že množina $\mathcal{B}$(**Osobní charakteristiky**) obsahuje téměř 1500 booleovských atributů, množina $\mathcal{B}$(**Tělesná aktivita**) 32 booleovských atributů a množina $\mathcal{B}$(**Měření**) 1175 booleovských atributů. To znamená, že v obr. 11.5 je zadáno více než 5.10^7 asociačních pravidel. Takto zadaný běh procedury 4ft-Miner trval 18 vteřin na PC s procesorem Intel(R) Core(TM)2Duo, 1,33 GHz a 2 GB RAM. Bylo verifikováno $0{,}51.10^6$ asociačních pravidel. Verifikaci ostatních zadaných pravidel bylo možné přeskočit, díky použitému algoritmu bylo zřejmé, že nemohou být pravdivá. Nebylo ovšem nalezeno žádné pravdivé asociační pravidlo.

Obvykle se v takové situaci snižují hodnoty parametrů 4ft-kvantifikátoru, byla tedy snížena hodnota parametru *Base* na 50 a byl použit 4ft-kvantifikátor $\Rightarrow_{0,9;50}$ místo $\Rightarrow_{0,9;100}$. Takto zadaný běh trval na stejném počítači 56 vteřin. Bylo verifikováno $1{,}72.10^6$ asociačních pravidel, vzhledem k nižší hodnotě parametru *Base* bylo možné přeskočit méně pravidel. Opět však nebylo nalezeno žádné pravdivé asociační pravidlo. Proto byla dále snížena hodnota obou parametrů – *Base* na 30 a *p* na 0,75. Příslušný běh trval 123 vteřin, bylo verifikováno $3{,}46.10^6$ asociačních pravidel a výsledkem bylo 274 prostých asociačních pravidel. Z těchto pravidel se 267 týkalo pouze atributu BMI, 7 pravidel se týkalo pouze atributu Cholesterol, žádné pravidlo se netýkalo obou atributů. Nejsilnější pravidlo, tedy pravidlo s největší konfidencí, vztahující se k atributu BMI, je pravidlo PacientAB $\Rightarrow_{0,86;37}$ BMI⟨23; 28⟩, kde

PacientAB = Vzdělání(vysokoškolské) ∧ Zodpovědnost(řídicí pracovník) ∧ Věk(45; 50⟩ ∧ Telaktza(převážně sedí, převážně stojí) ∧ Aktpozam(mírná).

Pravidlo říká, že v matici dat *Vstup* platí, že z pacientů, splňujících atribut – konjunkci – PacientAB, jich 86 % má BMI v intervalu (23; 28⟩, a že takových pacientů je v matici dat *Vstup* 37. Kontingenční tabulka pro toto pravidlo je uvedena v tab. 11.5.

Tabulka 11.5. Kontingenční tabulka 4ft(Pacient, BMI⟨23; 28⟩, *Vstup*)

Vstup	BMI⟨23; 28⟩	¬BMI⟨23; 28⟩
PacientAB	37	6
¬PacientAB	847	506

Je třeba poznamenat, že součet všech frekvencí uvedených v tab. 11.5 je 1396, tedy méně než počet řádků v matici dat *Vstup*, kterých je 1417. Je to proto, že byli vynecháni pacienti s chybějící informací, viz též odst. 11.2.3.

Nejsilnějším pravidlem, tedy pravidlem s největší konfidencí, týkající se atributu Cholesterol, je pravidlo Stav(rozvedený) ∧ Telaktza(převážně sedí) $\Rightarrow_{0,79;30}$ Cholesterol(< 240), které říká, že v matici dat *Vstup* platí, že z rozvedených pacientů, kteří v práci převážně sedí, jich 79 % má hodnotu cholesterolu menší než 240 a že takových pacientů je v matici dat *Vstup* 30. Kontingenční tabulka pro toto pravidlo je v tab. 11.6.

Tabulka 11.6. Kontingenční tabulka 4ft(Stav(rozvedený) ∧ Telaktza(převážně sedí), Cholesterol(< 240), *Vstup*)

Vstup	Cholesterol(< 240)	¬Cholesterol(< 240)
Stav(rozvedený) ∧ Telaktza(převážně sedí)	30	8
¬(Stav(rozvedený) ∧ Telaktza(převážně sedí))	780	579

Součet všech frekvencí v tab. 11.6 je 1397, tedy také méně než počet řádků v matici dat *Vstup*, protože byli vynecháni pacienti s chybějící informací.

Připomeňme, že výstupem GUHA procedury jsou všechna prostá tvrzení. Prosté tvrzení je takový relevantní vztah, který je pravdivý v analyzovaných datech a neplyne z jiného, jednoduššího, pravdivého a ve výstupu již uvedeného vztahu. V případě procedury 4ft-Miner se také pracuje s prostotou, její podrobnější výklad však přesahuje rámec této kapitoly. V případě tohoto příkladu prostota znamená, že pro atributy BMI a Cholesterol uvádíme pouze takové odvozené booleovské atributy, jejichž koeficient je nejkratší možný.

V případě pravidla Stav(rozvedený) ∧ Telaktza(převážně sedí) $\Rightarrow_{0,79;30}$ Cholesterol(< 240) platí, že z jeho pravdivosti plyne i pravdivost pravidla Stav(rozvedený) ∧ Telaktza(převážně sedí) $\Rightarrow_{0,79;30}$ Cholesterol(< 250). Je to proto, že z platnosti booleovského atributu Cholesterol(< 240) plyne pravdivost i booleovského atributu Cholesterol(< 250). Toto vyplývání booleovských atributů z jiného atributu souvisí právě s tím, že koeficient booleovského atributu Cholesterol(< 250) je delší než koeficient atributu Cholesterol(< 240). Nemá tedy smysl uvádět pravidla s delším koeficientem, než je nejkratší možný, jejich pravdivost je zřejmá, nejsou tudíž prostá.

Je třeba zdůraznit, že cílem tohoto odstavce je pouze naznačit možnosti procedury 4ft-Miner, cílem není věnovat se problematice interpretace výsledků. Přesto je však vhodné poznamenat, že množiny $\mathscr{B}$(***Osobní charakteristiky***), $\mathscr{B}$(***Tělesná aktivita***) i $\mathscr{B}$(***Měření***) jsou definovány tak, že obsahují rozumné booleovské charakteristiky skupin atributů ***Osobní charakteristiky***, ***Tělesná aktivita*** a ***Měření***, viz předchozí odstavec. Výraz rozumné je však třeba chápat v souvislosti se znalostí okolností sběru dat a významu kategorií jednotlivých atributů.

Jistě by bylo možné definovat například jiné kategorie pro atribut BMI s využitím známých intervalů hodnot. Pokud je hodnota BMI ≤ 18,5, znamená to podvýživu, interval (18,5; 25⟩ znamená normální váhu, interval (25; 30⟩ nadváhu atd. Z tohoto pohledu ale pravidlo Pacient $\Rightarrow_{0,86;37}$ BMI(23; 28⟩, říká, že pacienti splňující v matici dat *Vstup* konjunkci PacientAB mají v 86 % normální váhu nebo mírnější nadváhu, což jistě není nic překvapujícího. Problematika odfiltrování nezajímavých výsledků procedur DZD je důležitým výzkumným tématem, viz též oddíl 11.5.

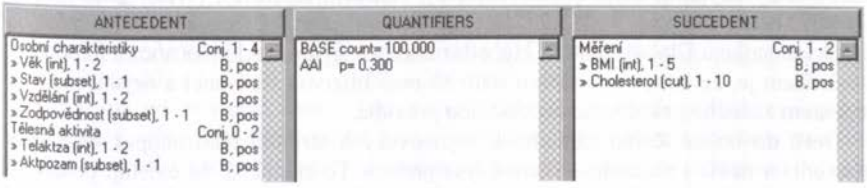
11.2.2.4 Bool(Osobní charakteristiky, Tělesné aktivity) $\Rightarrow^{+}_{p;Base}$ *Bool(Měření)*

Analytickou otázku Bool(O1B)

Vstup: Bool(***Osobní charakteristiky, Tělesné aktivity***) $\Rightarrow^{+}_{p;Base}$ Bool(***Měření***)

řešíme proto, abychom ukázali užitečnost kvantifikátoru $\Rightarrow^{+}_{p;Base}$. Chceme zjistit, pro které booleovské atributy φ, popisující kombinace osobních charakteristik a tělesných aktivit pacientů, a pro které booleovské atributy ψ, popisující výsledky měření v matici dat *Vstup*, platí, že relativní četnost pacientů splňujících ψ mezi pacienty splňujícími φ je alespoň o 100*p* procent větší než relativní četnost pacientů splňujících ψ v celé matici a že zároveň nejméně *Base* pacientů splňuje jak φ, tak i ψ.

Použijeme podobné zadání jako v obr. 11.5 pro otázku Bool(O1A). Jediný rozdíl je v tom, že použijeme 4ft-kvantifikátor $\Rightarrow^{+}_{0,3;100}$ místo 4ft-kvantifikátoru $\Rightarrow_{0,9;100}$, viz obr. 11.6.



Obr. 11.6 Zadání procedury 4ft-Miner pro otázku Bool(O1B) s 4ft-kvantifikátorem $\Rightarrow^{+}_{0,3;100}$.

To znamená, že opět je zadáno více než 5.10^7 asociačních pravidel. Běh procedury 4ft-Miner trval 18 vteřin a bylo verifikováno $0{,}51.10^6$ asociačních pravidel, verifikaci ostatních zadaných pravidel bylo možné přeskočit. Bylo nalezeno 16 prostých asociačních pravidel. Poznamenejme, že v případě kvantifikátoru $\Rightarrow^{+}_{p;Base}$ je každé pravdivé asociační pravidlo také prosté.

Nejsilnější pravidlo, tedy pravidlo odpovídající největšímu nárůstu relativní četnosti pacientů splňujících sukcedent, je pravidlo Věk(40; 50⟩ ∧ Zodpovědnost(ostatní) $\Rightarrow^{+}_{0,44;114}$ BMI(> 29). Kontingenční tabulka pro toto pravidlo je uvedena v tab. 11.7.

Tabulka 11.7. Kontingenční tabulka 4ft(Věk(40; 50⟩ ∧ Zodpovědnost(ostatní), BMI(> 29), *Vstup*)

Vstup	BMI(> 29)	¬BMI(> 29)
Věk(40; 50⟩ ∧ Zodpovědnost(ostatní)	114	401
¬(Věk(40;50⟩ ∧ Zodpovědnost(ostatní))	102	792

Platí, že relativní četnost pacientů splňujících sukcedent BMI(> 29) mezi pacienty splňujícími antecedent je $\frac{114}{114+401} = 0,22$ a relativní četnost pacientů splňujících sukcedent BMI(> 29) v celé matici je $\frac{114+102}{114+401+102+792} = 0,15$ a dále platí, že $0,22 = (1 + 0,44) . 0,15$.

Tedy pravidlo Věk(40; 50⟩ ∧ Zodpovědnost(ostatní) $\Rightarrow^{+}_{0,44;114}$ BMI(> 29) říká, že v matici dat *Vstup* platí, že relativní četnost pacientů s BMI větším než 29 mezi pacienty splňujícími Věk(40; 50⟩ ∧ Zodpovědnost(ostatní) je o 44 % větší než relativní četnost těchto pacientů v celé matici a takových pacientů je nejméně 114. Důležité je, že konfidence pravidla Věk(40; 50⟩ ∧ Zodpovědnost(ostatní) $\Rightarrow^{+}_{0,44;114}$ BMI(> 29) je pouze $\frac{114}{114+401} = 0,22$ a toto pravidlo není ve výstupu procedury 4ft-Miner při řešení úlohy Bool(O1A), viz předchozí odstavec. Přesto může být zajímavé z hlediska lékaře.

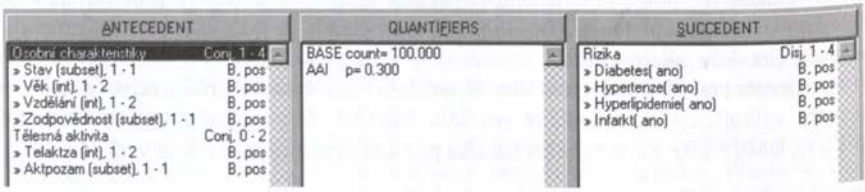
11.2.2.5 Bool(Osobní charakteristiky, Tělesné aktivity) $\Rightarrow^{+}_{p;Base}$ *Bool(Rizika)*

Analytickou otázku Bool(O2B)

Vstup: Bool(***Osobní charakteristiky***, ***Tělesné aktivity***) $\Rightarrow^{+}_{p;Base}$ Bool(***Rizika***)

řešíme proto, abychom ukázali užitečnost definice množiny $\mathscr{B}$(***Rizika***) booleovských charakteristik skupiny Rizika jako množiny disjunkcí délky 1–4 vytvořených ze základních booleovských atributů Diabetes(ano), Hypertenze(ano), Hyperlipidemie(ano) a Infarkt(ano), viz odst. 11.2.2.2. Jádrem problému je nízká frekvence základních booleovských atributů Diabetes(ano), Hypertenze(ano), Hyperlipidemie(ano) a Infarkt(ano). Důsledkem je, že konjunkce těchto atributů mají mizivou frekvenci a nejsou tudíž sukcedentem žádného pravdivého asociačního pravidla.

Delší disjunkce těchto základních booleovských atributů mají naopak dostatečnou frekvenci a navíc i rozumnou věcnou interpretaci. To znamená, že existují potenciálně věcně zajímavá a pravdivá asociační pravidla s těmito disjunkcemi v sukcedentu. Použijeme tedy podobné zadání, jaké vidíme na obr. 11.6 pro otázku Bool(O1B). Jediným rozdílem je to, že v sukcedentu použijeme definici množiny booleovských atributů $\mathscr{B}$(***Rizika***) místo množiny $\mathscr{B}$(***Měření***), viz obr. 11.7.



Obr. 11.7 Zadání procedury 4ft-Miner pro otázku Bool(O2B) s 4ft-kvantifikátorem $\Rightarrow^{+}_{0,3;100}$

Připomeňme, že množina $\mathscr{B}$(***Osobní charakteristiky***) obsahuje téměř 1500 booleovských atributů, množina $\mathscr{B}$(***Tělesná aktivita***) 32 booleovských atributů a množina

$\mathcal{B}$(**Rizika**) 15 booleovských atributů. V obrázku 11.7 je tedy zadáno $0{,}72.10^6$ relevantních asociačních pravidel. Běh procedury 4ft-Miner trval 1 vteřinu a bylo verifikováno 20 715 asociačních pravidel, verifikaci ostatních zadaných pravidel bylo možné přeskočit. Bylo nalezeno 7 prostých asociačních pravidel. Poznamenejme opět, že v případě kvantifikátoru $\Rightarrow^{+}_{p;Base}$ je každé pravdivé asociační pravidlo také prosté. Nejsilnější pravidlo, tedy pravidlo odpovídající největšímu nárůstu relativní četnosti pacientů splňujících sukcedent, je pravidlo

Věk(45; 55⟩ ∧ Telaktza(převážně sedí, převážně stojí) $\Rightarrow^{+}_{0,34;131}$ Hypertenze(ano) ∨ Hyperlipidemie(ano) ∨ Infarkt(ano).

Kontingenční tabulka pro toto pravidlo je znázorněna na tab. 11.8, píšeme HHIn(ano) místo disjunkce Hypertenze(ano) ∨ Hyperlipidemie(ano) ∨ Infarkt(ano).

Tabulka 11.8. Kontingenční tabulka 4ft(Věk(45; 55) ∧ (Telaktza(převážně sedí, převážně stojí), HHIn(ano), *Vstup*)

Vstup	HHIn(ano)	¬HHIn(ano)
Věk(45; 55⟩ ∧ Telaktza(převážně sedí, převážně stojí)	131	231
¬(Věk(45; 55⟩ ∧ Telaktza(převážně sedí, převážně stojí))	121	451

Platí, že mezi pacienty splňujícími antecedent Věk(45; 55⟩ ∧ Telaktza(převážně sedí, převážně stojí) je relativní četnost pacientů splňujících sukcedent Hypertenze(ano) ∨ Hyperlipidemie(ano) ∨ Infarkt(ano) rovna $\frac{131}{131+231}$ a platí $\frac{131}{131+231} = (1+0{,}34)\frac{131+121}{131+231+121+451}$, kde $\frac{131+121}{131+231+121+451}$ je relativní četnost pacientů splňujících sukcedent v celé matici. Tedy relativní četnost pacientů splňujících sukcedent mezi pacienty splňujícími antecedent je o 34 procent větší než mezi všemi pacienty popsanými v matici *Vstup*, přičemž vynecháváme pacienty s neúplnou informací. Pacienti, splňující sukcedent, jsou pacienty, jež mají alespoň jeden z rizikových faktorů hypertenze či hyperlipidemie aterosklerózy, nebo měli infarkt.

V sukcedentech všech sedmi asociačních pravidel se vyskytuje booleovský atribut Hypertenze(ano), u tří z nich v disjunkci s dalšími dvěma atributy a u čtyř s jedním dalším booleovským atributem. Ve výsledku se nevyskytuje žádný sukcedent délky 1. To je v souladu s tím, že pokud zadání modifikujeme tak, že použijeme v sukcedentu zadání `Rizika Conj, 1-4` místo `Rizika Disj, 1-4`, nedostaneme ve výsledku žádné asociační pravidlo. Podrobnější rozbor sedmi nalezených pravidel je zajímavý ale přesahuje rozsah této kapitoly. Cílem je pouze poukázat na význam disjunkce, v tomto případě v sukcedentu.

11.2.3 ASSOC a neúplná informace

V příkladech uvedených v předchozím odstavci, stejně jako v řadě praktických aplikací, se vyskytuje neúplná informace, údaje o hodnotách analyzovaných atributů pro některé objekty chybí. V souvislosti s metodou GUHA byl formulován specifický přístup ke zpracování neúplné informace – obecný princip zabezpečeného doplnění. Pokud jej

aplikujeme na asociační pravidla, lze říci, že asociační pravidlo $\varphi \approx \psi$ považujeme za pravdivé v matici M^X s neúplnou informací, pokud je pravdivé ve všech možných doplněních matice M^X. Tento fakt zapisujeme $Val(\varphi \approx \psi, M^X) = 1$. Doplněním matice dat M^X s neúplnou informací rozumíme každou matici takovou, ve které je každá z chybějících hodnot nahrazena některou z možných hodnot. Podobně považujeme asociační pravidlo $\varphi \approx \psi$ za nepravdivé v matici M^X s neúplnou informací, pokud je nepravdivé ve všech možných doplněních matice M^X, píšeme $Val(\varphi \approx \psi, M^X) = 0$. Pokud existuje doplnění matice M^X, ve kterém je asociační pravidlo $\varphi \approx \psi$ pravdivé, a zároveň existuje jiné doplnění matice M^X, v němž je asociační pravidlo $\varphi \approx \psi$ nepravdivé, pak říkáme, že hodnota $\varphi \approx \psi$ v matici M^X není známa, píšeme $Val(\varphi \approx \psi, M^X) = X$. V dalším výkladu jsou neformálně naznačeny potřebné podrobnosti.

V obrázku 11.8 je příklad matice M^X s neúplnou informací. Řádky matice odpovídají sledovaným objektům, sloupce atributům charakterizujícím jednotlivé objekty. Symbol X znamená chybějící informaci. Tedy pro objekt o_2 neznáme hodnotu atributů A a B, pro objekt o_3 neznáme hodnotu atributů B a C atd. Doplněním matice M^X se rozumí každá taková matice M, v níž je místo každého symbolu X dosazena některá z přípustných hodnot. Políčka matice, ve kterých je hodnota známa, se nemění. Podle principu zabezpečeného doplnění jsou určovány i hodnoty booleovských atributů odvozených ze sloupců matice.

To ale znamená, že atribut A(2) je v matici M^X pro objekt o_1 pravdivý, neboť v každé matici M, která je doplněním matice M^X, bude hodnota atributu A pro objekt o_1 rovna 2. Stejně tak bude hodnota atributu A pro objekt o_3 v každém doplnění 3, a tedy atribut A(2) je v matici M^X nepravdivý. Pro objekt o_2 ale jistě existuje doplnění M_1, v němž je hodnota atributu A pro objekt o_2 rovna 3, a tedy booleovský atribut A(2) je v doplnění M_1 nepravdivý. Existuje ale také doplnění M_2, ve kterém je hodnota atributu A pro objekt o_2 rovna 2, a tedy booleovský atribut A(2) je v doplnění M_2 pravdivý. V takovém případě tedy atribut A(2) podle principu zabezpečeného doplnění není ani pravdivý, ani nepravdivý. Říkáme také, že jeho hodnota je *nevím*, v obr. 11.8 je tato hodnota značena X.

Objekt	Atributy				Příklady booleovských (trojhodnotových) atributů vytvořených ze sloupců matice					
					Základní			Odvozené		
	A	B	C	...	A(2)	B(1,3)	C(4)	¬A(2)	B(1,3) ∧ C(4)	B(1,3) ∨ C(4)
o_1	2	1	4	...	1	1	1	0	1	1
o_2	X	X	4	...	*X*	*X*	1	*X*	*X*	1
o_3	3	X	X	...	0	*X*	*X*	1	*X*	*X*
...	...	...	...	...	...	...	...	...	...	...
o_{950}	1	2	X	...	0	0	*X*	1	0	*X*
o_{950}	2	4	4	...	1	0	1	0	0	1
o_{950}	x	5	3	...	*X*	0	0	*X*	0	0

Obr. 11.8 Příklad matice dat M^X s neúplnou informací.

V této souvislosti hovoříme o *trojhodnotové* logice s hodnotami *pravda*, *nepravda* a *nevím*. Atributy, které jsou v obr. 11.8 nazývány booleovskými, tedy vlastně nejsou booleovskými atributy v pravém slova smyslu, nazýváme je tak z důvodu návaznosti na obr. 11.1. Poněkud správnější je název *trojhodnotový* atribut uvedený v závorce, který budeme používat nadále. Hodnoty ostatních trojhodnotových atributů uvedených v obr. 11.8 se snadno určí pomocí principu zabezpečeného doplnění. K tomu slouží tabulky definující spojky $\neg$, $\wedge$ a $\vee$ pro trojhodnotovou logiku, viz obr. 11.9.

Asociační pravidlo $\varphi \approx \psi$ v matici M^X s neúplnou informací se tedy týká trojhodnotových atributů φ a ψ s hodnotami *pravda*, *nevím* a *nepravda*, které značíme 1, X a 0. To znamená, že vzájemný vztah atributů φ a ψ matici M^X je popsán devítipolní tabulkou $9ft(\varphi, \psi, M^X)$, viz tab. 11.9, místo čtyřpolní kontingenční tabulky dle tab. 11.1.

	$\neg$
1	0
X	X
0	1

$\wedge$	1	X	0
1	1	X	0
X	X	X	0
0	0	0	0

$\vee$	1	X	0
1	1	1	1
X	1	X	X
0	1	X	0

Obr. 11.9 Definice spojek $\neg$, $\wedge$ a $\vee$ pro trojhodnotovou logiku.

Tabulka 11.9. Devítipolní tabulka $9ft(\varphi, \psi, M^X)$ atributů φ a ψ v matici dat M^X s neúplnou informací

M^X	ψ	ψ_X	$\neg\psi$
φ	$f_{1,1}$	$f_{1,X}$	$f_{1,0}$
φ_X	$f_{X,1}$	$f_{X,X}$	$f_{X,0}$
$\neg\varphi$	$f_{0,1}$	$f_{0,X}$	$f_{0,0}$

Zde $f_{1,1}$ znamená počet řádků matice M^X, pro které jsou φ i ψ pravdivé, $f_{1,X}$ je počet řádků matice M^X, pro které je φ pravdivé a ψ má hodnotu *nevím* atd. Asociační pravidlo $\varphi \approx \psi$ považujeme podle principu zabezpečeného doplnění za pravdivé v matici M^X, pokud je pravdivé ve všech možných doplněních matice M^X. Pravidlo $\varphi \approx \psi$ považujeme za nepravdivé v M^X, pokud je nepravdivé ve všech možných doplněních matice M^X. Podle této definice bychom museli pro ověření pravdivosti $\varphi \approx \psi$ v M^X sestrojit všechna doplnění M' matice M^X, pro každé doplnění spočítat čtyřpolní tabulku $4ft(\varphi, \psi, M')$ a v ní ověřit podmínku danou 4ft-kvantifikátorem $\approx$. Každá taková čtyřpolní tabulka $4ft(\varphi, \psi, M')$ se ale dá odvodit z devítipolní tabulky $9ft(\varphi, \psi, M^X)$ způsobem naznačeným v tab. 11.10.

Tabulka 11.10 Čtyřpolní tabulka $4ft(\varphi, \psi, M')$ atributů φ a ψ v matici dat M', která je doplněním M^X

M'	ψ	$\neg\psi$
φ	$f_{1,1} + f_{1,X,a} + f_{X,1,a} + f_{X,X,a}$	$f_{1,0} + f_{1,X,b} + f_{X,0,b} + f_{X,X,b}$
$\neg\varphi$	$f_{0,1} + f_{X,1,c} + f_{0,X,c} + f_{X,X,c}$	$f_{0,0} + f_{0,X,d} + f_{X,0,d} + f_{X,X,d}$

Každý řádek matice M^X, pro který je φ pravdivé a ψ má hodnotu *nevím*, může být doplněn buď tak, že po doplnění je pro tento řádek ψ pravdivé, nebo tak, že po doplnění je ψ nepravdivé. Řádků, pro které je φ pravdivé a ψ má hodnotu *nevím* je $f_{1,X}$, viz tab. 11.9. Předpokládáme, že pro $f_{1,X,a}$ z nich je po doplnění ψ pravdivé a pro $f_{1,X,b}$ z nich je po doplnění ψ nepravdivé. To znamená, že doplněním řádků matice M^X, pro které je φ pravdivé a ψ má hodnotu *nevím* se v každém doplnění M' zvětší počet řádků, pro něž je φ i ψ pravdivé, o $f_{1,X,a}$ a počet řádků, pro které je φ pravdivé a ψ nepravdivé, se zvětší o $f_{1,X,b}$. Frekvence $f_{1,X}$ se tedy v doplnění M' rozdělí na frekvence $f_{1,X,a}$ a $f_{1,X,b}$, kde $f_{1,X} = f_{1,X,a} + f_{1,X,b}$. Podobně se rozdělí i frekvence $f_{X,1}$, $f_{X,0}$ a $f_{0,X}$. Frekvence $f_{X,X}$ se v doplnění M' rozdělí na čtyři frekvence: $f_{X,X,a}$, $f_{X,X,b}$, $f_{X,X,c}$ a $f_{X,X,d}$, viz tab. 11.10.

K ověření pravdivosti asociačního pravidla $\varphi \approx \psi$ v matici M^X je tedy zapotřebí ověřit platnost podmínky odpovídající 4ft-kvantifikátoru $\approx$ v každé čtyřpolní tabulce 4ft(φ, ψ, M') dle tab. 11.10. Pro důležité 4ft-kvantifikátory však lze ke každé devítipolní tabulce 9ft(φ, ψ, M^X) nalézt takovou čtyřpolní tabulku $\langle a_z, b_z, c_z, d_z\rangle$, pro niž platí, že podmínka odpovídající 4ft-kvantifikátoru $\approx$ je splněna ve čtyřpolní tabulce 4ft(φ, ψ, M') každého doplnění M' matice M^X, právě když je splněna v tabulce $\langle a_z, b_z, c_z, d_z\rangle$. Tabulku $\langle a_z, b_z, c_z, d_z\rangle$ nazýváme zabezpečeným doplněním tabulky 9ft(φ, ψ, M^X) pro 4ft-kvantifikátor $\approx$.

Příkladem zabezpečeného doplnění je zabezpečené doplnění pro 4ft-kvantifikátor $\Rightarrow_{p,Base}$ fundované implikace, který je definován podmínkou $\frac{a}{a+b} \geq p \wedge a \geq Base$, viz odst. 11.2.1. V tomto případě je zapotřebí aby a_z bylo co nejmenší a b_z co největší. Pokud totiž bude splněno $\frac{a_z}{a_z+b_z} \geq p \wedge a_z \geq Base$, tak pro každé jiné a', b' takové, že $a' \geq a_z$ a $b' \leq b_z$ je jistě splněno také $\frac{a'}{a'+b'} \geq p \wedge a' \geq Base$, neboť $\frac{a'}{a'+b'} \geq \frac{a_z}{a_z+b_z}$. Příklad takového zabezpečeného doplnění devítipolní tabulky 9ft(φ, ψ, M^X) z tab. 11.9 je uveden v tab. 11.11, předpokládáme, že se týká matice M_z.

Tabulka 11.11 Příklad zabezpečeného doplnění devítipolní tabulky 9ft(φ, ψ, M^X) uvedené v tab. 11.9

M_z	ψ	$\neg\psi$
φ	$f_{1,1}$	$f_{0,1} + f_{1,X} + f_{X,0} + f_{X,X}$
$\neg\varphi$	$f_{0,1} + f_{X,1} + f_{0,X}$	$f_{0,0}$

Způsob výpočtu zabezpečeného doplnění pro 4ft-kvantifikátor $\approx$ záleží na tom, do které třídy 4ft-kvantifikátorů kvantifikátor $\approx$ patří. 4ft-kvantifikátor $\Rightarrow_{p,Base}$ patří do třídy implikačních kvantifikátorů. Pro každý implikační kvantifikátor se zabezpečené doplnění počítá stejným způsobem jako pro kvantifikátor $\Rightarrow_{p,Base}$, tedy tak, aby a_z bylo co nejmenší a b_z co největší. Více informací o třídách kvantifikátorů najdeme v oddílu 11.4.

Procedura 4ft-Miner umožňuje zvolit pro vyhodnocování zabezpečené doplnění. V příkladech uvedených v předchozích odstavcích je však zvolena možnost vynechání neúplné informace. To znamená, že pro vyhodnocení asociačního pravidla $\varphi \approx \psi$ v mati-

ci M^X s devítipolní tabulkou 9ft(φ, ψ, M^X) dle tab. 11.9 se použije čtyřpolní tabulka $\langle f_{1,1}, f_{1,0}, f_{0,1}, f_{0,0}\rangle$. Tato čtyřpolní tabulka vznikne z devítipolní tabulky tak, že vynecháme z úvahy všechny řádky, v nichž se vyskytuje hodnota *nevím* pro některý z atributů φ, ψ. Další možností, kterou procedura 4ft-Miner poskytuje, je optimistické doplnění.

Zatímco v případě zabezpečeného doplnění považujeme asociační pravidlo $\varphi \approx \psi$ za pravdivé v matici M^X, pokud je pravdivé ve všech doplněních, v případě optimistického doplnění považujeme asociační pravidlo $\varphi \approx \psi$ za pravdivé, pokud je pravdivé alespoň v jednom doplnění. I při optimistickém doplnění se pro důležité 4ft-kvantifikátory můžeme vyhnout verifikaci ve všech možných doplněních. Je k dispozici předpis, jak pro danou devítipolní tabulkou 9ft(φ, ψ, M^X) nalézt takovou čtyřpolní tabulku $\langle a_o, b_o, c_o, d_o\rangle$, pro kterou platí, že podmínka odpovídající 4ft-kvantifikátoru $\approx$ je splněna ve čtyřpolní tabulce 4ft(φ, ψ, M') alespoň jednoho doplnění M' matice M^X, právě když je splněna v tabulce $\langle a_o, b_o, c_o, d_o\rangle$. Tabulku $\langle a_o, b_o, c_o, d_o\rangle$ nazýváme optimistickým doplněním tabulky 9ft(φ, ψ, M^X) pro 4ft-kvantifikátor $\approx$.

Příkladem optimistického doplnění je optimistické doplnění pro kvantifikátor $\Rightarrow_{p,Base}$ fundované implikace. V tomto případě je zapotřebí aby a_o bylo co největší a b_o co nejmenší, zdůvodnění je obdobné jako pro pesimistické doplnění pro tento kvantifikátor. Příklad takového optimistického doplnění devítipolní tabulky 9ft(φ, ψ, M^X) z tab. 11.9 je zaznamenán v tab. 11.12, předpokládáme, že se týká matice M_o. Poznamenejme, že tento způsob výpočtu optimistického doplnění je platný pro všechny implikační kvantifikátory.

Tabulka 11.12. Příklad zabezpečeného doplnění *devítipolní* tabulky 9ft(φ, ψ, M^X) uvedené v tab. 11.9

M_o	ψ	$\neg\psi$
φ	$f_{1,1} + f_{1,X} + f_{X,1} + f_{X,X}$	$f_{0,1}$
$\neg\varphi$	$f_{0,1}$	$f_{0,0} + f_{0,X} + f_{X,0}$

11.2.4 ASSOC a analýza nákupního košíku

Jak již bylo uvedeno, termín *asociační pravidlo* nebyl zaveden v souvislosti s metodou GUHA ale v souvislosti s analýzou dat o nákupních košících. V knize (Hájek a Havránek, 1978) je však v rámci definice a studií observačních kalkulů definován i pojem *asociační kvantifikátor*. Observační kalkul je speciální logický kalkul, jeho formule vypovídají o analyzovaných datech, viz oddíl 11.4. Lze říci, že asociační pravidla definovaná v práci (Agrawal a kol., 1996) jsou formulemi velmi jednoduchého observačního kalkulu. Některé další souvislosti jsou uvedeny v oddílu 11.4 podrobněji, viz např. (Rauch, 2013).

Dále naznačíme, jak lze asociační pravidla pro analýzu dat o nákupních košících vyjádřit pomocí booleovské matice dat a vhodného 4ft-kvantifikátoru. Data o obsahu nákupních košíků vznikají typicky na pokladnách v supermarketech. Ve zjednodušené formě si je lze představit jako tabulku dle obr. 11.10.

Každý řádek odpovídá jednomu nákupnímu košíku. V prvním sloupci je označení košíku, předpokládáme m košíků $k_1,\ldots, k_m$. V každém košíku se vyskytují položky zboží, například máslo, mléko, sýr, šunka, pivo atd. Předpokládáme, že supermarket prodává celkem n různých položek $p_1,\ldots, p_n$. Ve druhém sloupci tabulky dle obr. 11.10 je pro

každý košík uvedeno, které položky se v něm vyskytují. V košíku k_1 se vyskytují položky p_1, p_2, p_3 a p_5, stejné je to pro ostatní košíky.

k_1	p_1, p_2, p_3, p_5
k_2	p_2, p_3, p_4, p_{214}
...	...
k_{m-1}	$p_{12}, p_{94}, p_{214}, p_{249}, p_{12}, p_{94}, p_{214}, p_{215}\ p_{216}, p_{945}, p_{946}, p_{947}$
k_m	p_1, p_2, p_3, p_4, p_7

Obr. 11.10 Data o nákupních košících.

Asociačním pravidlem je výraz $X \rightarrow Y$, kde X a Y jsou disjunktní podmnožiny množiny všech položek (Agrawal a kol., 1996). Toto pravidlo říká, že nákupní košíky obsahující množinu položek X mají tendenci obsahovat i množinu položek Y. Příkladem může být pravidlo {párek, hořčice} $\rightarrow$ {pivo, chléb}, které říká, že košíky obsahující párek a hořčici obsahují často i pivo a chléb. Používají se dvě míry intenzity asociačního pravidla – *konfidence* a *podpora* (*support*), které jsou pro asociační pravidlo $X \rightarrow Y$ definovány takto:

$$\text{konfidence} = \frac{\text{počet košíků obsahujících } X \text{ i } Y}{\text{počet košíků obsahujících } X}$$

a

$$\text{podpora} = \frac{\text{počet košíků obsahujících } X \text{ i } Y}{\text{počet všech košíků}}.$$

Cílem úlohy dobývání asociačních pravidel je nalézt v daných datech všechna taková asociační pravidla, jejichž konfidence je větší než zadaný práh *conf* a jejichž podpora je větší než zadaný práh *supp*.

Tabulka 11.13. Data o nákupních košících ve formě matice dat Košíky

košík	atributy odpovídající položkám zboží						odvozené atributy	
	P_1	P_2	P_3	P_4	P_5	...	$P_1 \wedge P_2$	$P_3 \wedge P_4$
k_1	1	1	1	0	1	...	1	0
k_2	0	1	1	1	0	...	0	1
...	...	...	...	...	...	...	...	...
k_{m-1}	0	0	0	0	0	...	0	0
k_m	1	1	1	1	0	...	1	1

Tab. 11.14. Čtyřpolní tabulka 4ft($P_1 \wedge P_2$, $P_3 \wedge P_4$, Košíky)

Košíky	$P_3 \wedge P_4$	$\neg(P_3 \wedge P_4)$
$P_1 \wedge P_2$	a	b
$\neg(P_1 \wedge P_2)$	c	d

Asociační pravidlo $X \rightarrow Y$ můžeme definovat i pomocí matice dat, booleovských atributů, čtyřpolní tabulky a speciálního 4ft-kvantifikátoru. Definici naznačíme na příkladu pravidla $\{p_1, p_2\} \rightarrow \{p_3, p_4\}$. Data z obr. 11.10 lze zapsat ve formě matice dat Košíky, viz tab. 11.13. Každému nákupnímu košíku odpovídá jeden řádek matice dat, každé položce zboží pak jeden sloupec matice. Každé položce zboží odpovídá také jeden booleovský atribut, položce p_1 odpovídá atribut P_1 atd. Hodnoty atributů jsou ve sloupcích matice Košíky. Atribut P_1 je v řádku košíku k_1 pravdivý, protože košík k_1 obsahuje položku p_1, atribut P_1 je v řádku košíku k_2 nepravdivý, protože košík k_2 neobsahuje položku p_1 atd. Hodnota 1 znamená, že atribut je pravdivý a 0 značí, že atribut je nepravdivý. V tabulce 11.13 jsou i odvozené atributy $P_1 \wedge P_2$ a $P_3 \wedge P_4$. Atribut $P_1 \wedge P_2$ říká, že košík obsahuje položku p_1 i položku p_2, atribut $P_3 \wedge P_4$ říká, že košík obsahuje položku p_3 i položku p_4. V tabulce 11.14 je čtyřpolní tabulka 4ft($P_1 \wedge P_2$, $P_3 \wedge P_4$, Košíky).

Platí tedy, že a je počet košíků, které obsahují množinu položek $\{p_1, p_2\}$ i množinu $\{p_3, p_4\}$, $a + b$ je počet košíků, které obsahují množinu položek $\{p_1, p_2\}$ a $a + b + c + d$ je počet všech košíků. Lze definovat 4ft-kvantifikátor $\rightarrow_{conf,supp}$ tak, že mu přiřadíme podmínku $\frac{a}{a+b} \geq conf \wedge \frac{a}{a+b+c+d} \geq supp$. Úlohu nalézt v daných datech všechna taková asociační pravidla, jejichž konfidence je větší než zadaný práh *conf* a jejichž podpora je větší než zadaný práh *supp*, lze potom řešit pomocí procedury ASSOC, vstupem je matice dat dle tab. 11.14, kvantifikátor $\rightarrow_{conf,supp}$ a vhodně zadané množiny relevantních antecedentů a sukcedentů. Poznamenejme, že kvantifikátor $\rightarrow_{conf,supp}$ je implementován v proceduře 4ft-Miner.

11.3 Systém LISp-Miner

Úloha nalézt v daných datech o nákupních košících všechna taková asociační pravidla, jejichž konfidence je větší než zadaný práh *conf* a jejichž podpora je větší než zadaný práh *supp*, se obvykle řeší pomocí *apriori*, viz odst. 11.1. Jeho podstata spočívá v tom, že nejprve najdeme všechny množiny položek, jejichž podpora je větší než zadaný práh *supp* a poté vygenerujeme pravidla, jejichž konfidence je větší než zadaný práh *conf*. Při hledání množiny položek, jejíž podpora je větší než zadaný práh *supp*, se využívá faktu, že podpora každé nadmnožiny X' dané množiny položek X nemůže být větší než podpora množiny X.

Algoritmus *apriori* se však v proceduře ASSOC nepoužívá. Implementace procedury ASSOC je založena na reprezentaci analyzovaných dat pomocí vhodných bitových řetízků (Rauch, 1978). Znamená to, že každý základní booleovský atribut A(a), jehož koeficient je dán jedinou kategorií, je reprezentován bitovým řetízkem. Každému řádku analyzované matice odpovídá jeden bit. V bitu je 1, pokud je atribut A(a) pravdivý v odpovídajícím řádku, v opačném případě je v tomto bitu 0. To znamená, že například základní booleovský atribut Město(Plzeň) je v matici dle obr. 11.1 reprezentován bitovým řetízkem, v prvním bitu je 1, ve druhém 0 atd. Všechny další booleovské atributy, které jsou zadány vstupem procedury ASSOC, jsou také reprezentovány booleovskými

řetízky, které se vytváří pomocí rychlých bitových operací. Frekvence v kontingenční tabulce také počítáme pomocí bitových operací včetně operace pro rychlý výpočet počtu jedniček v daném řetízku. Například frekvence a v tabulce 4ft(φ, ψ, M) je rovna počtu jedniček v bitovém řetízku reprezentujícím konjunkci $\varphi \wedge \psi$.

Při implementaci procedury 4ft-Miner (Šimůnek, 2003), (Rauch, Šimůnek, 2005a) byl vyvinut systém podprogramů pro práci s bitovými řetízky, jenž umožnil implementaci dalších GUHA procedur určených pro získávání složitějších vztahů, než jsou asociační pravidla. Všechny tyto procedury jsou integrovány do systému LISp-Miner (Šimůnek, 2010). Tento systém byl od počátku vyvíjen jako systém pro práci s daty uchovávanými v databázových systémech, je však možné zpracovat i matice dat v jiné formě. Příkladem GUHA procedury pro získávání složitějších vztahů, než jsou asociační pravidla, je procedura SD4ft-Miner, její možnosti jsme naznačili v odst. 11.3.1. Přehled ostatních GUHA procedur implementovaných v systému LISp-Miner se nalézá v odst. 11.3.2.

11.3.1 GUHA procedura SD4ft-Miner

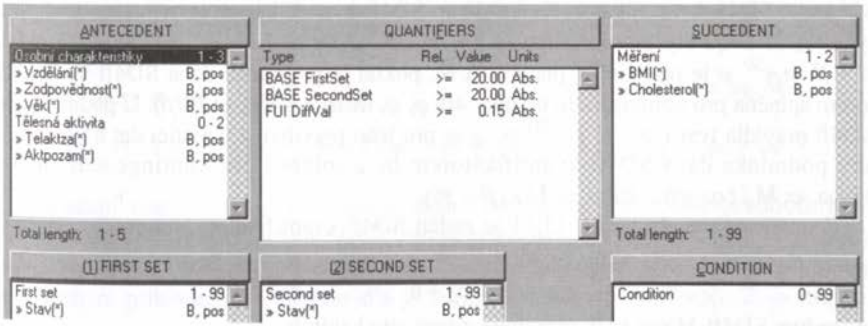
V odstavci 11.2.2 byla řešena analytická otázka: „Pro které booleovské atributy φ popisující kombinace osobních charakteristik a tělesných aktivit pacientů a pro které booleovské atributy ψ popisující výsledky měření v matici dat *Vstup* platí, že alespoň 100p procent z pacientů splňujících φ splňuje i ψ a zároveň nejméně *Base* pacientů splňuje jak φ, tak i ψ?“. Byla použita procedura 4ft-Miner se zadáním dle obr. 11.5 ale s kvantifikátorem $\Rightarrow_{0,75;30}$. Výsledkem bylo 274 prostých asociačních pravidel. Nejsilnější pravidlo mělo konfidenci 0,86. Jednoduchou analýzou lze zjistit, že z nalezených pravidel se jich 126 týkalo atributu Stav, z nich 123 se jich týkalo základního atributu Stav(ženatý) a tři zbylá základního atributu Stav(rozvedený).

Naskýtá se tedy přirozená otázka: „Pro které dvojice rodinného stavu je v matici dat *Vstup* největší rozdíl mezi vztahem booleovských atributů φ a ψ, kde φ popisuje kombinace osobních charakteristik a tělesných aktivit pacientů a ψ udává výsledky měření v matici dat *Vstup*?“. Lze také říci, že nás zajímají asociační pravidla $\varphi \Rightarrow_{p;Base} \psi$ a dvojice základních booleovských atributů Stav(u) a Stav(v), kde u ≠ v, u i v patří mezi stavy {ženatý, rozvedený, svobodný, vdovec}, takové, že rozdíl konfidencí podmíněných asociačních pravidel $\varphi \Rightarrow_{p;Base} \psi$ / Stav(u) a $\varphi \Rightarrow_{p;Base} \psi$ / Stav(v) v matici dat *Vstup* je větší než zadaná mez.

Podmíněné asociační pravidlo $\varphi \Rightarrow_{p;Base} \psi$ / Stav(u) se v matici dat *Vstup* vyhodnocuje jako asociační pravidlo $\varphi \Rightarrow_{p;Base} \psi$ v matici dat *Vstup* / Stav(u). Matice dat *Vstup* / Stav(u) je matice, která vznikne z matice dat *Vstup* tak, že ponecháme pouze ty řádky, které splňují atribut Stav(u), a ostatní vynecháme. Podobně postupujeme pro matici dat *Vstup* / Stav(v). Například matice dat *Vstup* / Stav(ženatý) tedy obsahuje pouze ty řádky z původní matice, které se týkají ženatých mužů. Lze také říci, že základní booleovské atributy Stav(u) a Stav(v) určují dvě podmnožiny množiny řádků matice dat *Vstup*.

Procedura SD4ft-Miner je určena pro řešení úloh jako je úloha naznačená v předchozím textu. Poznamenejme, že písmena SD pocházejí z anglického Sets Difference. Na obrázku 11.11 je uveden příklad zadání procedury SD4ft-Miner pro řešení naznačené úlohy.

Hledáme dvojice asociačních pravidel $\varphi \Rightarrow_{p;Base} \psi$ / Stav(u) a $\varphi \Rightarrow_{p;Base} \psi$ / Stav(v), kde množina relevantních antecedentů φ je dána stejně jako na obr. 11.5 (odst. 11.2.2.3) až na to, že mezi osobními charakteristikami se nevyskytuje atribut Stav. Množina přípustných sukcedentů ψ je dána stejně jako v obr. 11.5. Dále jsou v částech FIRST SET a SECOND SET zadány naznačeným způsobem páry podmnožin řádků matice *Vstup* pomocí základních booleovských atributů Stav(u) a Stav(v). V části QUANTIFIERS je požadováno, aby v první množině bylo alespoň 20 řádků splňujících antecedent φ i sukcedent ψ, viz text `Base FirstSet >= 20,00 Abs`, podobný postup zvolíme pro druhou množinu. Text `FUI DiffVal >= 0,15 Abs` znamená, že požadujeme, aby konfidence pravidla $\varphi \Rightarrow_{p,Base} \psi$ / Stav(u) byla alespoň o 0,15 větší než konfidence pravidla $\varphi \Rightarrow_{p;Base} \psi$ / Stav(v).



Obr. 11.11 Příklad zadání procedury SD4ft-Miner.

Takto zadaný běh procedury SD4ft-Miner trval 8 minut a 35 vteřin, bylo verifikováno $16{,}1.10^6$ dvojic podmíněných asociačních pravidel, 67 dvojic splnilo zadané podmínky. Největší rozdíl konfidencí byl 0,257 a týkal se pravidel s atributem Cholesterol v sukcedentu. Druhý největší rozdíl konfidencí byl 0,232 a týkal se pravidel s atributem BMI v sukcedentu. Dvojici pravidel vypovídajících o BMI uvádíme jako příklad. Jsou to pravidla VzZd $\Rightarrow_{0,456;21}$ BMI(≤ 24) / Stav(rozvedený) a VzZd $\Rightarrow_{0,224;74}$ BMI(≤ 24) / Stav(ženatý), kde VzZd = Vzdělání(základní, vyučen) ∧ Zodpovědnost(ostatní). Lze říci, že mezi rozvedenými pacienty splňujícími VzZd je 45,6 % mužů s BMI ≤ 24, zatímco mezi ženatými jich je pouze 22,4 %, tedy dvakrát méně. Připomeňme, že BMI ≤ 18,5 znamená podvýživu, BMI v intervalu (18,5; 25⟩ znamená normální váhu a že analyzovaná matice dat se týká pouze mužů. Kontingenční tabulky pro obě pravidla jsou uvedeny na obr. 11.12.

Vstup / Stav(rozvedený)	BMI(≤ 24)	¬BMI(≤ 24)
VzZd	21	25
¬VzZd	13	45

4ft(VzZd, BMI(≤ 24), *Vstup* / Stav(rozvedený))

Vstup / Stav(ženatý)	BMI(≤ 24)	¬BMI(≤ 24)
VzZd	74	256
¬VzZd	223	654

4ft(VzZd, BMI(≤ 24), *Vstup* / Stav(rozvedený))

Obr. 11.12 4ft-tabulky pro VzZd $\Rightarrow_{0,456;21}$ BMI(≤ 24) / Stav(rozvedený)) a VzZd $\Rightarrow_{0,224;74}$ BMI(≤ 24) / Stav(ženatý)).

Uvedenou dvojici asociačních pravidel píšeme také jako jeden výraz

$$\text{Stav(rozvedený)} \times \text{Stav(ženatý): VzZd} \Rightarrow^{SD}_{0,232;21;74} \text{BMI(}\leq 24\text{)},$$

symbol $\Rightarrow^{SD}_{0,232;21;74}$ se nazývá SD4ft-kvantifikátor, celý výraz se nazývá SD4ft pravidlo. Číslo 0,232 je rozdíl konfidencí obou pravidel, 21 značí počet objektů splňujících antecedent i sukcedent prvního pravidla a 74 vyjadřuje počet objektů splňujících antecedent i sukcedent druhého pravidla.

Máme k dispozici celou řadu SD4ft-kvantifikátorů, lze pracovat i s podmíněnými pravidly SD4ft. Obecně má SD4ft pravidlo tvar $\alpha \times \beta$: $\varphi \approx^{SD} \psi$, podmíněné SD4ft pravidlo má tvar $\alpha \times \beta$: $\varphi \approx^{SD} \psi / \chi$. Zde α, β, φ, ψ a χ jsou booleovské atributy a symbol $\approx^{SD}$ značí SD4ft-kvantifikátor. Ke každému SD4ft-kvantifikátoru je přiřazena podmínka týkající se dvou čtyřpolních tabulek $\langle a_1, b_1, c_1, d_1\rangle$ a $\langle a_2, b_2, c_2, d_2\rangle$. SD4ft pravidlo $\alpha \times \beta$: $\varphi \approx^{SD} \psi$ je pravdivé v matici dat M, pokud je podmínka daná SD4ft-kvantifikátorem splněna pro kontingenční tabulky 4ft(φ, ψ, M/α) a 4ft(φ, ψ, M/β). U podmíněného SD4ft pravidla tvaru $\alpha \times \beta$: $\varphi \approx^{SD} \psi / \chi$ se pro jeho pravdivost v matici dat M požaduje, aby podmínka daná SD4ft-kvantifikátorem byla splněna pro kontingenční tabulky 4ft(φ, ψ, M / ($\alpha \wedge \chi$)) a 4ft(φ, ψ, M / ($\beta \wedge \chi$)).

Poznamenejme, že v obr. 11.11 je zadán SD4ft-kvantifikátor, kterému je přiřazena podmínka $a_1 \geq 20 \wedge a_2 \geq 20 \wedge \frac{a_1}{a_1 + b_1} - \frac{a_2}{a_2 + b_2} \geq 0{,}15$. Podrobnější a přesnější popis procedury SD4ft-Miner však přesahuje rámec této kapitoly.

11.3.2 Další GUHA procedury systému LISp-Miner

Prozatím jsme uvedli dvě GUHA procedury implementované v systému LISp-Miner – proceduru 4ft-Miner a proceduru SD4ft-Miner. Stručně naznačíme možnosti dalších pěti GUHA procedur implementovaných v systému LISp-Miner. Jde o procedury AC4ft-Miner, KL-Miner, SDKL-Miner, CF-Miner a SDCF-Miner.

AC4ft-Miner. GUHA procedura AC4ft-Miner hledá v dané matici dat SD4ft pravidlo. AC4ft pravidlo je výraz $\varphi_{St} \wedge \Phi_{Zm} \approx^{Ac} \psi_{St} \wedge \Psi_{Zm}$, kde φ_{St} a ψ_{St} jsou booleovské atributy, $\approx^{Ac}$ vyjadřuje AC4ft-kvantifikátor. Výraz Φ_{Zm} se nazývá změna antecedentu a výraz Ψ_{Zm} změna sukcedentu. Změna antecedentu Φ_{Zm} je vytvořena ze změn základních booleovských atributů, stejně jako antecedent asociačního pravidla vznikne ze základních booleovských atributů, podobně postupujeme pro změnu sukcedentu Ψ_{Zm}.

Změnou základního booleovského atributu je výraz A($\alpha_P \rightarrow \alpha_K$), kde α_P a α_K jsou disjunktní podmnožiny kategorií atributu **A**. Základní booleovský atribut A(α_P) se nazývá *počáteční stav atributu* A, základní booleovský atributu A(α_K) pak *koncový stav atributu* A. Pro každou změnu antecedentu Φ_{Zm} lze sestrojit *počáteční stav antecedentu* φ_P tak, že každou změnu základního booleovského atributu A($\alpha_P \rightarrow \alpha_K$) nahradíme počátečním stavem A(α_P). Podobně sestrojíme *koncový stav antecedentu* φ_K tak, že každý výraz A($\alpha_P \rightarrow \alpha_K$) nahradíme *koncovým stavem* A(α_K). *Počáteční stav sukcedentu* ψ_P a *koncový stav sukcedentu* ψ_K sestrojíme analogicky. Atribut φ_{St} se nazývá *stabilní*

část antecedentu a atribut ψ_{St} *stabilní část sukcedentu. Počátečním stavem pro AC4ft pravidlo* $\varphi_{St} \wedge \Phi_{Zm} \approx^{Ac} \psi_{St} \wedge \Psi_{Zm}$ je asociační pravidlo $\varphi_{St} \wedge \varphi_P \approx_P \psi_{St} \wedge \psi_P$ a koncovým stavem pro toto SD4ft pravidlo je asociační pravidlo $\varphi_{St} \wedge \varphi_K \approx_K \psi_{St} \wedge \psi_K$.

Ke každému AC4ft-kvantifikátoru $\approx^{Ac}$ je přiřazena podmínka týkající se dvou čtyřpolních tabulek $\langle a_P, b_P, c_P, d_P \rangle$ a $\langle a_K, b_K, c_K, d_K \rangle$. AC4ft pravidlo $\varphi_{St} \wedge \Phi_{Zm} \approx^{Ac} \psi_{St} \wedge \Psi_{Zm}$ je pravdivé v matici dat M, pokud je podmínka daná Ac4ft-kvantifikátorem splněna pro kontingenční tabulky 4ft($\varphi_{St} \wedge \varphi_P$, $\psi_{St} \wedge \psi_P$, M) a 4ft($\varphi_{St} \wedge \varphi_K$, $\psi_{St} \wedge \psi_K$, M). Jde o kontingenční tabulky pro počáteční stav $\varphi_{St} \wedge \varphi_P \approx_P \psi_{St} \wedge \psi_P$ a pro koncový stav $\varphi_{St} \wedge \varphi_K \approx_K \psi_{St} \wedge \psi_K$, viz obr. 11.13.

M	$\psi_{St} \wedge \psi_P$	$\neg(\psi_{St} \wedge \psi_P)$
$\varphi_{St} \wedge \varphi_P$	a_P	b_P
$\neg(\varphi_{St} \wedge \varphi_P)$	c_P	d_P

4ft($\varphi_{St} \wedge \varphi_P$, $\psi_{St} \wedge \psi_P$, M)

M	$\psi_{St} \wedge \psi_K$	$\neg(\psi_{St} \wedge \psi_K)$
$\varphi_{St} \wedge \varphi_K$	a_K	b_K
$\neg(\varphi_{St} \wedge \varphi_K)$	c_K	d_K

4ft($\varphi_{St} \wedge \varphi_K$, $\psi_{St} \wedge \psi_K$, M)

Obr. 11.13 4ft-tabulky pro pravidla $\varphi_{St} \wedge \varphi_P \approx_P \psi_{St} \wedge \psi_P$ a $\varphi_{St} \wedge \varphi_K \approx_K \psi_{St} \wedge \psi_K$.

Příkladem změny základního booleovského atributu je výraz Stav(svobodný → ženatý), který říká, že se pacient oženil. Jiným příkladem je výraz Pivo(≥ 2 → nepije), který říká, že pacient, který pil více než dvě piva denně, úplně přestal pít pivo. Velmi jednoduchým příkladem AC4ft pravidla je výraz Věk(40; 45⟩ ∧ Pivo(≥ 2 → nepije) $\Rightarrow^{AC}_{0,2;30;30}$ BMI(> 26). Jeho počátečním stavem je asociační pravidlo Věk(40; 45⟩ ∧ Pivo(≥ 2) $\Rightarrow_{0,57;32}$ BMI(> 26), koncovým stavem je Věk(40; 45⟩ ∧ Pivo(nepije) $\Rightarrow_{0,37;39}$ BMI (> 26). Toto AC4ft pravidlo naznačuje, že pokud muži ve věku 40–45 let, kteří pijí více než dvě piva denně, přestanou pít pivo, sníží se podíl mužů s BMI větší než 26 z 57 % na 37 %.

Procedura AC4ft-Miner byla inspirována akčními pravidly navrženými v práci (Ras, Wieczorkowska, 2000). Vstupem pro proceduru AC4ft-Miner jsou, kromě AC4ft-kvantifikátoru, i zadání šesti množin booleovských atributů (cedentů), které určují přípustné atributy pro *stabilní část antecedentu*, *počáteční stav antecedentu*, *koncový stav antecedentu*, *stabilní část sukcedentu*, *počáteční stav sukcedentu* a *koncový stav sukcedentu*. Prozatím nevelké zkušenosti naznačují, že AC4ft pravidla a procedura AC4ft-Miner mohou být užitečnými nástroji (Rauch, Šimůnek, 2009a), jsou však stále předmětem vývoje.

Podobně nevelké ale slibné jsou zkušenosti s dalšími GUHA procedurami, tj. KL-Miner, SDKL-Miner, CF-Miner a SDCF-Miner (Rauch, Šimůnek, 2005a), (Hájek a kol., 2010). Všechny pracují jak s booleovskými, tak i s obecnými kategoriálními atributy. Možnosti těchto procedur jsou stručně naznačeny dále. Předpokládáme, že ρ a σ jsou atributy – sloupce matice M. Kategorie (přípustné hodnoty) atributu ρ jsou $r_1,\ldots, r_K$ a kategorie atributu σ jsou $c_1,\ldots, c_L$. KL-tabulkou $KL(\rho, \sigma, M)$ atributů ρ a σ v matici dat M dle obr. 11.1 je matice $K \times L$ celých čísel $n_{i,j}$, viz obr. 11.14. Číslo $n_{i,j}$ znamená počet řádků matice M, pro které je hodnota atributu ρ rovna r_i a hodnota atributu σ je c_j. CF-tabulkou $CF(\rho, M)$ atributu ρ v matici dat M dle obr. 11.14 je vektor K celých

čísel $n_1,\ldots, n_K$, viz obr. 11.14. Číslo n_i znamená počet řádků matice M, pro které hodnota atributu ρ je r_i.

M	c_1	...	c_L
r_1	$n_{1,1}$	⋯	$n_{1,L}$
⋮	⋮	⋱	⋮
r_K	$n_{K,1}$	⋯	$n_{K,L}$

KL-tabulka $KL(\rho, \sigma, \mathsf{M})$

M	r_1	⋯	r_K
	n_1	...	n_K

CF-tabulka $CF(\rho, \mathsf{M})$

Obr. 11.14 KL-tabulka $KL(\rho, \sigma, \mathsf{M})$ a CF-tabulka $CF(\rho, \mathsf{M})$ v matici dat M.

Procedury KL-Miner, SDKL-Miner, CF-Miner a SDCF-Miner využívají prostředky pro práci s bitovými řetízky vytvořené při implementaci procedury 4ft-Miner (Rauch, Šimůnek, 2005a). Podstatou implementace těchto procedur je rychlý výpočet tabulek $KL(\rho, \sigma, \mathsf{M}/\chi)$ a CF-tabulka $CF(\rho, \mathsf{M}/\chi)$. Připomeňme, že M/χ je matice dat, která vznikne z matice dat M tak, že vynecháme všechny řádky nesplňující booleovský atribut χ (pro jednoduchost zde předpokládáme, že existuje řádek splňující χ). Například frekvence $n_{1,1}$ z tabulky $KL(\rho, \sigma, \mathsf{M})$ se spočítá jako počet jedniček v konjunkci bitových řetízků reprezentujících kategorie r_1 a c_1, podobně obdobná frekvence $n_{1,1}$ z tabulky $KL(\rho, \sigma, \mathsf{M}/\chi)$ se spočítá jako počet jedniček v konjunkci tří bitových řetízků reprezentujících kategorie r_1 a c_1 a booleovský atribut χ (Rauch a kol., 2005).

GUHA procedura KL-Miner hledá v dané matici dat KL-pravidla. KL-pravidlo je výraz $\rho \approx^{KL} \sigma / \chi$, kde ρ a σ jsou kategoriální atributy, χ je booleovský atribut a $\approx^{KL}$ značí KL-kvantifikátor. Ke každému KL-kvantifikátoru $\approx^{KL}$ je přiřazena podmínka týkající se každé z KL-tabulek uvedených na obr. 11.14. KL-pravidlo $\rho \approx^{KL} \sigma/\chi$ je pravdivé v matici dat M, pokud je podmínka daná KL-kvantifikátorem $\approx^{KL}$ splněna pro KL-tabulku $KL(\rho, \sigma, \mathsf{M} / \chi)$. KL-pravidlo $\rho \approx^{KL} \sigma / \chi$ tedy říká, že pokud je splněna podmínka daná booleovským atributem χ, pak kategoriální atributy ρ a σ jsou v dané matici dat ve vztahu daném KL-kvantifikátorem $\approx^{KL}$. První zkušenosti s procedurou KL-Miner jsou popsány v článcích (Lín a kol., 2004), (Rauch a kol., 2005).

GUHA procedura CF-Miner hledá v dané matici dat CF-pravidla. CF-pravidlo je výraz $\approx^{CF} \rho / \chi$, kde ρ je kategoriální atribut, χ představuje booleovský atribut a $\approx^{CF}$ značí CF-kvantifikátor. Ke každému CF-kvantifikátor $\approx^{CF}$ je přiřazena podmínka týkající se všech CF-tabulek znázorněných na obr. 11.14. CF-pravidlo $\approx^{CF} \sigma / \chi$ je pravdivé v matici dat M, pokud je podmínka daná CF-kvantifikátorem $\approx^{CF}$ splněna pro CF-tabulku $CF(\rho, \mathsf{M} / \chi)$. CF-pravidlo $\approx^{CF} \rho / \chi$ tedy říká, že podmíněné frekvence jednotlivých kategorií atributu splňují podmínku danou CF-kvantifikátorem $\approx^{CF}$. Podmíněnými frekvencemi se rozumí frekvence, které se týkají pouze řádků dané matice splňujících podmínku danou booleovským atributem χ.

SDKL-Miner. GUHA procedura SDKL-Miner hledá v dané matici dat SDKL-pravidla. SDKL-pravidlo je výraz $\alpha \times \beta$: $\rho \approx^{SDKL} \sigma / \chi$, kde ρ a σ jsou kategoriální atributy, α, β a χ značí booleovské atributy a $\approx^{SDKL}$ je SDKL-kvantifikátor. Ke každému SDKL-kvantifikátoru $\approx^{SDKL}$ je přiřazena podmínka týkající se každé dvojice KL-tabu-

lek. SDKL-pravidlo $\alpha \times \beta$: $\rho \approx^{SDKL} \sigma / \chi$ je pravdivé v matici dat M, pokud je podmínka daná SDKL-kvantifikátorem $\approx^{SDKL}$ splněna pro dvojici KL-tabulek $KL(\rho, \sigma, M / (\chi \wedge \alpha))$ a $KL(\rho, \sigma, M / (\chi \wedge \beta))$. SDKL-pravidlo $\alpha \times \beta$: $\rho \approx^{SDKL} \sigma / \chi$ tedy říká, že pokud je splněna podmínka daná booleovským atributem χ, pak se vztah kategoriálních atributů ρ a σ na podmnožinách objektů daných booleovskými atributy α a β liší způsobem daným SDKL-kvantifikátorem $\approx^{SDKL}$. Podmnožina objektů daná booleovským atributem α odpovídá řádkům dané matice, pro které je splněn atributem α, podobná úvaha platí pro podmnožinu objektů danou booleovským atributem β.

SDCF-Miner. GUHA procedura SDCF-Miner hledá v dané matici dat SDCF-pravidla. SDCF-pravidlo je výraz $\alpha \times \beta$: $\approx^{SDCF} \rho / \chi$, kde ρ je kategoriální atribut, α, β a χ vyjadřují booleovské atributy a $\approx^{SDCF}$ značí SDCF-kvantifikátor. Ke každému SDCF-kvantifikátoru $\approx^{SDCF}$ je přiřazena podmínka týkající se každé dvojice CF-tabulek. SDCF-pravidlo $\alpha \times \beta$: $\approx^{SDCF} \rho / \chi$ je pravdivé v matici dat M, pokud je podmínka daná SDCF-kvantifikátorem $\approx^{SDCF}$ splněna pro dvojici CF-tabulek $CF(\rho, M / (\chi \wedge \alpha))$ a $CF(\rho, M / (\chi \wedge \beta))$. SDCF-pravidlo $\alpha \times \beta$: $\approx^{SDCF} \rho / \chi$ tedy říká, že pokud je splněna podmínka daná booleovským atributem χ, pak podmíněné frekvence kategoriálního atributu ρ se na podmnožinách objektů daných booleovskými atributy α a β liší způsobem daným SDCF-kvantifikátorem $\approx^{SDCF}$.

11.4 Observační kalkuly

Mezi deseti nejdůležitějšími problémy výzkumu v DZD je jako první uvedena nutnost vytvořit jednotnou teorii pro data mining (Qiang, Xindong, 2006). Teoretické základy metody GUHA byly vytvořeny v rámci širší teorie automatické tvorby hypotéz. Matematické základy této teorie jsou obsahem monografie (Hájek, Havránek, 1978). Lze je chápat jako příspěvek k jednotné teorii pro data mining, jejich principy jsou naznačeny v odstavci 11.4.1.

Důležitou roli v nich hrají observační kalkuly. Jsou to speciální logické kalkuly, jejichž formule odpovídají tvrzením o analyzovaných datech. Příkladem jsou observační kalkuly asociačních pravidel, jejich formule odpovídají asociačním pravidlům. Základní informace o observačních kalkulech asociačních pravidel najdeme v odstavci 11.4.2. Byla dosažena řada pro praktické použití důležitých a teoreticky zajímavých výsledků týkajících se těchto kalkulů. Většina z nich se váže ke třídám 4ft-kvantifikátorů. Přehled důležitých tříd 4ft-kvantifikátorů uvádíme v odstavci 11.4.3. Mezi důležité výsledky týkající se observačních kalkulů asociačních pravidel patří dedukční pravidla. Stručný úvod k těmto dedukčním pravidlům je v odstavci 11.4.4.

11.4.1 Matematické základy automatické tvorby hypotéz

Monografie (Hájek, Havránek, 1978) se zabývá odpověďmi na dvě základní otázky:

- Mohou počítače formulovat a zdůvodnit vědecké hypotézy?
- Mohou počítače porozumět empirickým datům a zpracovat je racionálním způsobem s využitím prostředků moderní matematické logiky a statistiky s cílem vytvořit racionální obraz pozorovaného empirického světa?

Odpověď na tyto otázky vychází ze schématu induktivní inference:

$$\frac{\text{teoretické předpoklady, observační tvrzení}}{\text{teoretické tvrzení}}.$$

To znamená, že když přijmeme vhodné teoretické předpoklady a ověříme vhodné tvrzení o napozorovaných datech, tak přijmeme závěr – teoretické tvrzení. Podstatné je, že k teoretickému tvrzení vede až vhodné tvrzení o datech, ne ovšem data samotná.

Uvedené schéma induktivní inference vede v knize (Hájek, Havránek, 1978) k formulaci dalších otázek:

(L0) – V jakých jazycích jsou formulována observační a teoretická tvrzení? (Jaká je syntax a sémantika těchto jazyků? Jaký je jejich vztah ke klasickému predikátovému počtu prvního řádu?)

(L1) – Jaká jsou racionální induktivní inferenční pravidla umožňující přejít od observačních tvrzení k tvrzením teoretickým? Co to znamená, že teoretické tvrzení je zdůvodněno?

(L2) – Existují racionální metody pro rozhodnutí, zda je teoretické tvrzení na základě daných teoretických předpokladů a observačních tvrzení ověřeno?

(L3) – Za jakých podmínek je teoretické tvrzení nebo množina teoretických tvrzení zajímavá vzhledem k dané úloze vědeckého poznání?

(L4) – Existují metody pro návrh takové množiny tvrzení, která je tak zajímavá, jak je to jen možné?

O uvedených otázkách je v monografii (Hájek, Havránek, 1978) detailně diskutováno a také na ně odpovídáno. V souladu s touto monografií lze odpovědi na tyto otázky pokládat za matematické základy automatické tvorby hypotéz.

Racionální induktivní inferenční pravidla umožňující přejít od observačních tvrzení k teoretickým vycházejí ze statistických přístupů. Jsou využity odhady různých parametrů a testy statistických hypotéz. Teoretické tvrzení je pokládáno za zdůvodněné, pokud je splněna podmínka týkající se odhadu použitého parametru, nebo pokud je tvrzení odpovídající testu statistické hypotézy splněno v daných observačních datech. Důležité je, že mezi teoretickými a observačními tvrzeními je vztah 1 : 1.

Odpověď na otázku (L0) vede mimo jiné k definici observační kalkulů. Observačními kalkuly jsou logické kalkuly, jejichž formule odpovídají tvrzením verifikovaným při ověřování, zda teoretické tvrzení je zdůvodněné. Příkladem jsou observační predikátové kalkuly. Neformálně řečeno, observační predikátové kalkuly vzniknou z klasických predikátových kalkulů tak, že se připustí pouze konečné modely a přidají se zobecněné kvantifikátory. Konečné modely odpovídají observačním datům, která jsou vždy konečná. Zobecněné kvantifikátory umožňují vyjádřit podmínky ohledně odhadu použitého parametru nebo tvrzení odpovídajících testu statistické hypotézy. Kniha (Hájek, Havránek, 1978) se podrobně věnuje monadickým observačním predikátovým kalkulům, jejichž jednoduché modifikace vedou k logickým kalkulům asociačních pravidel.

11.4.2 Observační kalkuly asociačních pravidel

Důležitým observačním kalkulem je monadický predikátový observační kalkul, který vznikne modifikacemi klasického monadického predikátového kalkulu. Příkladem kla-

sického monadického predikátového kalkulu je predikátový kalkul s monadickými predikáty $P_1,\ldots, P_n$. Klasickými kvantifikátory jsou $\forall$ a $\exists$. Výrazy $P_1(x)$, $P_2(y)$, $P_1(x) \wedge P_2(y)$ jsou příklady otevřených formulí tohoto kalkulu, výrazy $(\exists x)P_1(x)$ a $(\forall x)(P_1(x) \wedge P_2(x))$ představují příklady uzavřených formulí. Formule se mohou týkat nekonečných struktur, například bodů v rovině nebo přirozených čísel.

Z takového kalkulu vznikne monadický observační predikátový kalkul tak, že přidáme zobecněné kvantifikátory a omezení na struktury, v nichž se formule interpretují. Příklady zobecněných kvantifikátorů jsou 4ft-kvantifikátory: $\Rightarrow_{p,Base}$, $\Leftrightarrow_{p,Base}$, $\equiv_{p,Base}$ a $\Rightarrow^{+}_{p,Base}$ uvedené v odstavci 11.2.1. Výraz $(\Rightarrow_{p,Base} x)(P_1(x), P_2(y))$ je příklad otevřené formule s volnou proměnnou y. Výrazy $(\Rightarrow_{p,Base} x)(P_1(x), P_2(x))$ a $(\Rightarrow^{+}_{p,Base} x)(P_1(x) \wedge P_2(x), P_3(x) \vee P_4(x))$ ukazují příklady uzavřených formulí. Formule observačních kalkulů interpretujeme v konečných strukturách, v případě monadického observačního kalkulu s predikáty $P_1,\ldots, P_n$ je takovou strukturou matice dat s hodnotami 1 (pravda) a 0 (nepravda), viz tab. 11.13.

V monografii (Hájek, Havránek, 1978) je uvedena formální definice observačních predikátových kalkulů včetně definice hodnot otevřených i uzavřených formulí. Neformálně lze říci, že hodnotu uzavřené formule $(\Rightarrow_{p,Base} x)(P_1(x), P_2(x))$ v matici dat M určujeme pomocí čtyřpolní tabulky $4ft(P_1, P_2, M)$ a podmínky $\frac{a}{a+b} \geq p \wedge a \geq Base$ týkající se takových čtyřpolních tabulek, viz odst. 11.2.1. Je-li tato podmínka splněna, je formule $(\Rightarrow_{p,Base} x)(P_1(x), P_2(x))$ pravdivá v matici dat M atd. Podobně určujeme hodnotu uzavřené formule $(\Rightarrow^{+}_{p,Base} x)(P_1(x) \wedge P_2(x), P_3(x) \vee P_4(x))$ v matici dat M pomocí čtyřpolní tabulky $4ft(P_1 \wedge P_2, P_3 \vee P_4, M)$. Autoři knihy (Hájek, Havránek, 1978) dosáhli řady významných výsledků, které se týkají teoretických vlastností observačních kalkulů, jako je jejich rozhodnutelnost nebo definovatelnost zobecněných kvantifikátorů. Získali i důležité výsledky ohledně dedukčních pravidel týkajících se formulí se zobecněnými kvantifikátory. Některé výsledky jsou vázány na třídy zobecněných kvantifikátorů. V uvedené knize jsou definovány a studovány třídy asociačních a implikačních 4ft-kvantifikátorů.

V pracích (Rauch, 1998a), (Rauch, 1998b), (Rauch, 2005), (Rauch, 2013) jsou definovány a studovány logické kalkuly asociačních pravidel. Formulemi těchto kalkulů jsou asociační pravidla tvaru $\varphi \approx \psi$, jejichž příklady najdeme v oddílu 11.2. Poznamenejme, že každé takové asociační pravidlo lze získat z uzavřené formule $(\approx x)(\varphi(x), \psi(x))$ monadického observačního kalkulu, kde $\varphi(x)$ a $\psi(x)$ jsou odvozené predikáty – booleovské kombinace výrazů $P_1(x),\ldots, P_n(x)$ vytvořené pomocí spojek $\vee$, $\wedge$ a $\neg$. Nejprve vynecháme všechny výrazy (x) a formuli píšeme ve tvaru formule $\varphi \approx \psi$. Poté nahradíme každý predikát $P_1,\ldots, P_n$ atomickým booleovským atributem $A(\alpha)$, viz odst. 11.2.1. Asociační pravidla tvaru $\varphi \approx \psi$ lze také získat podobnými transfomacemi z formulí kalkulů s kvalitativními hodnotami (calculi with qualitative values) definovaných v knize (Hájek, Havránek, 1978).

Pro kalkuly asociačních pravidel bylo dosaženo nových teoretických výsledků, které rozšiřují výsledky uvedené v monografii (Hájek, Havránek, 1978). Jejich přehled je v práci (Rauch, 2013). Nové výsledky souvisí s třídami 4ft-kvantifikátorů, bylo definováno a studováno několik nových tříd 4ft-kvantifikátorů.

11.4.3 Třídy 4ft-kvantifikátorů

Jak jsme již uvedli, termín *asociační pravidlo* nebyl zaveden v souvislosti s metodou GUHA, v monografii (Hájek, Havránek, 1978) je však definován pojem *asociační kvantifikátor*, třída asociačních kvantifikátorů a také třída implikačních kvantifikátorů.

4ft-kvantifikátor $\approx$ je *implikační*, jestliže pro každou dvojici čtyřpolních tabulek $\langle a, b, c, d\rangle$ a $\langle a', b', c', d'\rangle$ plyne z faktu $\approx(a, b, c, d) = 1$ a z platnosti $a' \geq a \wedge b' \leq b$, že $\approx(a', b', c', d') = 1$. Podmínka $a' \geq a \wedge b' \leq b$ se nazývá *podmínka zachování pravdivosti pro implikační kvantifikátory*. Tato podmínka říká, že čtyřpolní tabulka $\langle a', b', c', d'\rangle$ je *lepší vzhledem k implikaci* než čtyřpolní tabulka $\langle a, b, c, d\rangle$ (*i-better*), viz (Hájek, Havránek, 1978).

Tvrzení $a' \geq a \wedge b' \leq b$ znamená, že je-li $\langle a, b, c, d\rangle$ čtyřpolní tabulka φ a ψ v matici dat M a jestliže $\langle a', b', c', d'\rangle$ je čtyřpolní tabulka φ a ψ v matici dat M′, pak v matici M′ je více řádků splňujících jak φ, tak i ψ než v matici M a zároveň, že v M′ je méně řádků splňujících φ a nesplňujících ψ než v M. Tedy, jestliže $a' \geq a \wedge b' \leq b$, pak je rozumné očekávat, že je-li implikační pravidlo $\varphi \approx \psi$ (tj. asociační pravidlo vyjadřující implikaci pomocí implikačního kvantifikátoru $\approx$) pravdivé v matici M, pak je také pravdivé v matici M′. Oprávněnost tohoto očekávání je pro implikační kvantifikátory zajištěna právě uvedenou definicí. Je snadné dokázat, že kvantifikátor $\Rightarrow_{p,Base}$ fundované implikace je implikační.

V knize (Hájek, Havránek, 1978) je dokázáno, že kvantifikátor $\Rightarrow^{!}_{p,Base}$ dolní kritické implikace je také implikační. Tento kvantifikátor je definován podmínkou $\sum_{i=a}^{a+b} \frac{(a+b)!}{i!(a+b-i)!} p^i (1-p)^{a+b-i} \leq \alpha \wedge a \geq Base$. Je založen na testu nulové hypotézy, tj. že podmíněná pravděpodobnost sukcedentu za podmínky antecedentu je menší nebo rovna p oproti alternativní hypotéze, že je větší než p. Jde o test na hladině významnosti α (Hájek a kol., 1976).

Tabulka 11.16. Příklady tříd 4ft-kvantifikátorů definovaných pomocí podmínky zachování pravdivosti

Třída 4ft-kvantifikátorů	Podmínka zachování pravdivosti	Příklad 4ft-kvantifikátoru	
		symbol	podmínka pravdivosti
implikační	$\mathrm{TPC}_{\Rightarrow}$: $a' \geq a \wedge b' \leq b$	$\Rightarrow_{p,Base}$	$\frac{a}{a+b} \geq p \wedge a \geq Base$
dvojitě imlikační	$\mathrm{TPC}_{\Leftrightarrow}$: $a' \geq a \wedge b' \leq b \wedge c' \leq c$	$\Leftrightarrow_{p,Base}$	$\frac{a}{a+b+c} \geq p \wedge a \geq Base$
Σ-dvojitě imlikační	$\mathrm{TPC}_{\Sigma,\Leftrightarrow}$: $a' \geq a \wedge b' + c' \leq b + c$		
ekvivalenční (asociační)	$\mathrm{TPC}_{\equiv}$: $a' \geq a \wedge b' \leq b \wedge c' \leq c \wedge d' \geq d$	$\equiv_{p,Base}$	$\frac{a+d}{a+b+c+d} \geq p \wedge a \geq Base$
Σ-ekvivalenční	$\mathrm{TPC}_{\Sigma,\equiv}$: $a' + d' \geq a + d \wedge b' + c' \leq b + c$		

Existuje řada dalších tříd 4ft-kvantifikátorů, mnoho z nich lze definovat pomocí vhodné podmínky zachování pravdivosti. Každá podmínka zachování pravdivosti se týká dvou čtyřpolních tabulek $\langle a, b, c, d\rangle$ a $\langle a', b', c', d'\rangle$. Příkladem je podmínka za-

chování pravdivosti pro implikační kvantifikátory $a' \geq a \wedge b' \leq b$, kterou značíme $TPC_{\Rightarrow}(a, b, c, d, a', b', c', d')$, zkráceně $TPC_{\Rightarrow}$. Je-li třída Ω 4ft-kvantifikátorů definována pomocí podmínky zachování pravdivosti $TPC_{\Omega}(a, b, c, d, a', b', c', d')$, pak to znamená, že 4ft-kvantifikátor $\approx$ patří do třídy Ω, pokud z faktu $\approx (a, b, c, d)$ a z platnosti $TPC_{\Omega}(a, b, c, d, a', b', c', d')$ plyne, že také $\approx (a', b', c', d') = 1$. Příklady důležitých tříd 4ft-kvantifikátorů definovaných pomocí podmínky zachování pravdivosti vidíme v tab. 11.16 spolu s 4ft-kvantifikátory, které do těchto tříd patří.

Třída **dvojitě implikačních kvantifikátorů** je definována v práci (Rauch, 1998a). Využity jsou úvahy uvedené v článku (Hájek a kol., 1983). Důvod pro tuto definici lze vidět i v analogii s výrokovým počtem. Jestliže u a v jsou výroky a jestliže jak $u \rightarrow v$, tak i $v \rightarrow u$ jsou pravdivé, pak u je ekvivalentní s v (symbol $\rightarrow$ zde značí výrokovou spojku „implikace"). Můžeme se tedy pokusit vyjádřit ekvivalenci atributů φ a ψ pomocí „dvojitě implikačního kvantifikátoru" $\Leftrightarrow$ tak, že $\varphi \Leftrightarrow \psi$ platí, právě když platí jak $\varphi \Rightarrow^{*} \psi$, tak i $\psi \Rightarrow^{*} \varphi$, kde $\Rightarrow^{*}$ je vhodný implikační kvantifikátor.

Jestliže aplikujeme podmínku $TPC_{\Rightarrow}$ zachování pravdivosti pro implikační kvantifikátory na $\varphi \Rightarrow^{*} \psi$, dostaneme $a' \geq a \wedge b' \leq b$. Jestliže ji aplikujeme na $\psi \Rightarrow^{*} \varphi$, dostaneme $a' \geq a \wedge c' \leq c$, ($c$ je zde místo b, viz tab. 11.1). Spojením dostaneme podmínku zachování pravdivosti pro dvojitě implikační kvantifikátory $a' \geq a \wedge b' \leq b \wedge c' \leq c$.

Kvantifikátor $\Leftrightarrow_{p,Base}$ dvojité fundované implikace je nejen dvojitě implikační ale i Σ-dvojitě implikační. Třída **Σ-dvojitě implikačních kvantifikátorů** definovaná v práci (Rauch, 1998a) je důležitou podtřídou třídy dvojitě implikačních kvantifikátorů, platí pro ni důležité teoretické výsledky, viz odst. 11.4.4.

Důvodem pro definici dvojitě implikačního kvantifikátoru je analogie s výrokovým počtem, kdy vyjadřujeme ekvivalenci výroků v a u pomocí dvojice implikací $u \rightarrow v$ a $v \rightarrow u$. Ekvivalenci výroků v a u lze však vyjádřit i pomocí dvojice implikací $u \rightarrow v$ a $\neg u \rightarrow \neg v$. Lze se tedy pokusit vyjádřit ekvivalenci atributů φ a ψ i pomocí „**ekvivalenčního kvantifikátoru**" $\equiv^{*}$ tak, že $\varphi \equiv^{*} \psi$ platí, právě když platí jak $\varphi \Rightarrow^{*} \psi$, tak i $\neg\varphi \Rightarrow^{*} \neg\psi$, kde $\Rightarrow^{*}$ je vhodný implikační kvantifikátor. Jestliže aplikujeme podmínku zachování pravdivosti pro implikační kvantifikátory na $\varphi \Rightarrow^{*} \psi$, dostaneme $a' \geq a \wedge b' \leq b$. Jestliže ji aplikujeme na $\neg\varphi \Rightarrow^{*} \neg\psi$, obdržíme $d' \geq d \wedge c' \leq c$. Spojením získáme podmínku zachování pravdivosti pro ekvivalenční kvantifikátory $a' \geq a \wedge b' \leq b \wedge c' \leq c \wedge d' \geq d$.

Definice ekvivalenčního kvantifikátoru je převzata z knihy (Hájek, Havránek, 1978), v níž se takto definovaný kvantifikátor nazývá *asociační* (associational). Zde však budeme i nadále používat termín „ekvivalenční kvantifikátor" místo „asociační kvantifikátor" abychom se vyhnuli možným nedorozuměním s (i když později definovaným, ale obecně vžitým) pojmem „asociační pravidlo", viz např. (Agrawal a kol., 1996). Poznamenejme ještě, že dle práce (Rauch, 1998a) není v naprosté většině případů asociační pravidlo použité v práci (Agrawal a kol., 1996) ekvivalenční, a tudíž ani asociační dle citované monografie (Hájek, Havránek, 1978).

Kvantifikátor $\equiv_{p,Base}$ fundované ekvivalence je nejen ekvivalenční, ale i **Σ-ekvivalenční**. Třída Σ-ekvivalenčních kvantifikátorů definovaná v práci (Rauch, 1998a) je důležitou podtřídou třídy dvojitě implikačních kvantifikátorů.

Je také definována a studována řada dalších tříd asociačních kvantifikátorů, viz (Rauch, 2013). V monografii (Hájek, Havránek, 1978) je definována třída **symetrických**

4ft-kvantifikátorů tak, že 4ft-kvantifikátor $\approx$ je symetrický, pokud platí $\approx(a, b, c, d) = \approx(a, c, b, d)$.

11.4.4 Dedukční pravidla pro asociační pravidla

Důležité výsledky ohledně logických kalkulů asociačních pravidel se týkají dedukčních pravidel tvaru $\frac{\varphi \approx \psi}{\varphi' \approx \psi'}$, kde $\varphi \approx \psi$ a $\varphi' \approx \psi'$ jsou asociační pravidla. Je-li takové dedukční pravidlo korektní, pak z pravdivosti $\varphi \approx \psi$ v matici dat M plyne pravdivost $\varphi' \approx \psi'$ v M. Příkladem velmi jednoduchého korektního dedukčního pravidla je pravidlo $\frac{A(a_1) \Rightarrow_{0,9;100} B(b_1)}{A(a_1) \Rightarrow_{0,9;100} B(b_1, b_2)}$, které říká, že když je v dané matici dat pravdivé asociační pravidlo $A(a_1) \Rightarrow_{0,9;100} B(b_1)$, tak je v ní pravdivé i pravidlo $A(a_1) \Rightarrow_{0,9;100} B(b_1, b_2)$. Korektnost tohoto dedukčního pravidla plyne z faktu, že když řádek matice splňuje $B(b_1)$, tak splňuje i $B(b_1, b_2)$.

Takováto dedukční pravidla mohou být použita například k redukci výstupu procedury ASSOC. Jestliže asociační pravidlo $\varphi \approx \psi$ je součástí výstupu (tedy je pravdivé v analyzované matici) a jestliže $\frac{\varphi \approx \psi}{\varphi' \approx \psi'}$ je korektní dedukční pravidlo, pak již není nutné, aby asociační pravidlo $\varphi' \approx \psi'$ bylo součástí výstupu. Použité dedukční pravidlo musí však být dostatečně jednoduché z pohledu uživatele, viz též (Chrz, 2007). Jiná možnost použití dedukčních pravidel se týká snížení počtu skutečně testovaných pravidel. Jeli asociační pravidlo $\varphi \approx \psi$ pravdivé v analyzované matici a $\frac{\varphi \approx \psi}{\varphi' \approx \psi'}$ je korektní dedukční pravidlo, pak již není třeba testovat asociační pravidlo $\varphi' \approx \psi'$. Další možnost použití dedukčních pravidel se týká filtrování důsledků známých faktů. Ta je v současné době předmětem výzkumu, viz odst. 11.5.

Z uvedeného textu plyne, že je důležité znát korektní dedukční pravidla tvaru $\frac{\varphi \approx \psi}{\varphi' \approx \psi'}$. Existuje teoretický výsledek, který umožňuje převést otázku, zda $\frac{\varphi \approx \psi}{\varphi' \approx \psi'}$ je korektní dedukční pravidlo, na otázku, zda jisté formule výrokového počtu jsou nebo nejsou tautologiemi. Tento výsledek je k dispozici pro důležité podtřídy tříd implikačních, Σ-dvojitě implikačních a Σ-ekvivalenčních kvantifikátorů a některé další třídy 4ft-kvantifikátorů, viz (Rauch, 2005), (Rauch, 2008), (Rauch, 2010) a (Rauch, 2013). Naznačíme, jak tento výsledek vypadá pro implikační kvantifikátory.

Nejprve budeme definovat třídu zajímavých implikačních kvantifikátorů. Poznamenejme, že pro každý implikační kvantifikátor $\Rightarrow^*$ je hodnota $\Rightarrow^*(a, b, c, d)$ pro každou čtyřpolní tabulku $\langle a, b, c, d\rangle$ nezávislá na c i d. Pro implikační kvantifikátor píšeme tedy pouze $\Rightarrow^*(a, b)$ místo $\Rightarrow^*(a, b, c, d)$. Říkáme, že *implikační kvantifikátor* $\Rightarrow^*$ *je a-závislý*, jestliže existují nezáporná celá čísla a, a', b tak, že $\Rightarrow^*(a, b) \neq \Rightarrow^*(a', b)$. Analogicky definujeme *b-závislý implikační kvantifikátor*. Říkáme, že *implikační kvantifikátor* $\Rightarrow^*$ *je*

zajímavý, pokud je *a*-závislý, *b*-závislý a pokud platí $\Rightarrow^*(0,0)$. V práci (Rauch, 1986) je dokázáno, že pro „regulární" hodnoty parametrů jsou všechny implikační kvantifikátory implementované v GUHA proceduře 4ft-Miner zajímavé. Dále je zde dokázána i věta, která říká, že pro každé dedukční pravidlo $\frac{\varphi \Rightarrow^* \psi}{\varphi' \Rightarrow^* \psi'}$ s implikačním kvantifikátorem $\Rightarrow^*$ existují formule Φ_{1A}, Φ_{1B} a Φ_2 výrokového počtu takové, že dedukční pravidlo $\frac{\varphi \Rightarrow^* \psi}{\varphi' \Rightarrow^* \psi'}$ je korektní, právě když platí:

1. Φ_{1A} i Φ_{1B} jsou tautologie výrokového počtu

nebo

2. Φ_2 je tautologie výrokového počtu.

Popis konstrukce formulí Φ_{1A}, Φ_{1B} a Φ_2 však přesahuje rozsah této kapitoly.

11.5 Související výzkum

Dobývání znalostí z databází je nejen široce aplikovanou disciplínou informatiky, ale i oblastí, v níž probíhá intenzivní výzkum a vývoj. Přehled deseti nejdůležitějších výzev pro výzkum v DZD najdeme v článku (Qiang, Xindong, 2006), mnohé z nich vycházejí ze zkušeností z aplikací. Metoda GUHA byla vícekrát aplikována na analýzu reálných dat, viz např. (Hájek, 1978), (Hájek, 1981), (Berka a kol., 2009). Zajímavé jsou i aplikace na reálných datech prováděné v rámci výuky DZD (Rauch, Tomečková, 2007). Poznamenejme, že metodologii aplikací metody GUHA uvedenou v práci (Hájek a kol., 1983) lze považovat za jednu z prvních metodologií pro DZD. Z aplikací metody GUHA vyplývají obdobné požadavky na další výzkum jako z aplikací ostatních metod DZD.

Zkušenosti s aplikacemi ukazují, v souladu s článkem (Qiang, Xindong, 2006), že je nezbytné využívat specifických doménových znalostí. Stejně důležité je věnovat větší pozornost prezentaci výsledků uživateli. Výzkum související s metodou GUHA je proto věnován plnění těchto požadavků. Práce s doménovými znalostmi využívá observační kalkuly, formalizované doménové znalosti lze vyjádřit pomocí vhodných formulí observačních kalkulů. Užitečným prostředkem pro prezentaci výsledků uživateli se ukazují analytické zprávy. Jádrem takové analytické zprávy jsou formule observačních kalkulů.

Krom toho, že s aplikacemi metody GUHA jsou spojeny stejné výzvy pro výzkum jako s aplikacemi ostatních metod DZD, objevuje se řada podnětů jak vylepšit specifické prostředky metody GUHA. Patří sem například teoretické i implementační problémy spojené s aplikací fuzzy přístupů nebo využití gridů pro zvýšení výpočtové síly GUHA procedur.

Zmíněné podněty nastartovaly práce na několika dílčích, vzájemně provázaných výzkumných tématech. Jejich přehled je uveden v odstavci 11.5.1. Koordinace řešení některých výzkumných úloh spolu s využitím nástrojů znalostního inženýrství umožňují komplexnější řešení problémů spojených s aplikacemi doménových znalostí a prezentací výsledků uživateli. Výsledkem je výzkumný projekt SEWEBAR, který je stručně představen v odstavci 11.5.2.

11.5.1 Výzkumná témata

Současný rozvoj metody GUHA souvisí zejména s těmito výzkumnými tématy:
- získáním dalších zkušenosti s aplikacemi jednotlivých procedur GUHA,
- studiem vzájemné závislostí výsledků jednotlivých procedur,
- shromážděním a využitím relevantních doménových znalostí,
- automatizací běhu GUHA procedur,
- využitím gridových technologií,
- rozvojem observačních kalkulů,
- GUHA procedurami pro multirelační data mining,
- aplikací fuzzy přístupů.

Většina z těchto témat souvisí se dvěma současnými implementacemi metody GUHA – systémem LISp-Miner (Šimůnek, 2005), (Rauch, Šimůnek, 2005a), (Rauch, Šimůnek, 2005b) a systémem Ferda Data Miner (Ralbovský, Kuchař, 2007). Některá z témat ale navazují na dřívější nápady a záměry. Uveďme krátké komentáře k jednotlivým tématům.

1. Máme k dispozici poměrně **rozsáhlé zkušenosti s aplikacemi** procedury 4ft-Miner, které jsou využívány při dalších aplikacích a jež jsou i východiskem pří výzkumu možností automatizace běhu jednotlivých GUHA procedur. Důležitá část zkušeností s procedurou 4ft-Miner se váže k zadávání booleovských cedentů a dílčích cedentů (pojem uvedený v odst. 11.2.2.1). Tyto zkušenosti využijeme i při práci s dalšími procedurami, neboť booleovské cedenty jsou důležitou součástí všech implementovaných GUHA procedur. Dílčí zkušenosti máme i s procedurami SD4ft-Miner (Rauch, Šimůnek, 2009b), Ac4ft-Miner (Rauch, Šimůnek, 2009a) a KL-Miner (Rauch a kol., 2005), (Lín a kol., 2004). S ostatními procedurami (SDKL-Miner, CF-Miner a SDCF-Miner) však dostatek zkušeností není. Jejich získání a využití je jedním z výzkumných témat.

Výsledky jednotlivých GUHA procedur se navzájem doplňují. Například pomocí procedury KL-Miner lze získat informaci, že v matici dat M je za podmínky dané booleovským atributem χ pozitivní závislost mezi kategoriálními atributy ρ a σ v tom smyslu, že když rostou hodnoty ρ (např. denní spotřeba piva pacienta), tak rostou i hodnoty σ (např. BMI). Tato informace odpovídá KL-vztahu $\rho \approx^{KL} \sigma/\chi$ s Kendallovým kvantifikátorem, příklady takových závislostí jsou v příspěvcích (Rauch a kol., 2005), (Lín a kol., 2004). Pokud v matici dat M nalezneme dostatečně silné asociační pravidlo $\varphi \wedge \rho(malé) \Rightarrow_{p,Base} \sigma(velké)$ nebo $\varphi \wedge \rho(malé) \Rightarrow^{+}_{p,Base} \sigma(velké)$, lze to chápat jako indicii, že za podmínky dané booleovským atributem φ neplatí zmíněná pozitivní závislost mezi kategoriálními atributy ρ a σ. Předpokládáme že $\rho(malé)$ a $\sigma(velké)$ jsou booleovské atributy automaticky vytvořené procedurou 4ft-Miner na základě vhodného zadání. Výraz $\rho(malé)$ znamená, že hodnota atributu ρ je malá, podobně $\sigma(velké)$ vyjadřuje, že hodnota σ je velká. Impulsem pro použití procedury 4ft-Miner může být právě výsledek procedury KL-Miner.

2. Takových **závislostí mezi GUHA procedurami** je jistě více. Jednotlivé závislosti nejsou příliš složité, jejich přehledné zmapování a využití je však rozsáhlým výzkumným tématem. Výsledky lze využít například pro vytvoření systému vzájemně provázaných analytických otázek i pro filtrování výstupů. Některé přístupy a úvahy použité při řešení závislostí mezi jednotlivými procedurami lze také využít při zkoumání možností využití doménových znalostí, viz následující výklad.

3. Velkým problémem DZD je **využití doménových znalostí**. Příkladem doménové znalosti je všeobecně přijímaný fakt, že pokud u pacientů roste denní spotřeba piva, tak roste i jejich BMI. To lze vyjádřit symbolicky například výrazem *pivo* ↑↑ *BMI*. Takovouto doménovou znalost lze využít například pro odfiltrování výsledků procedury 4ft-Miner, které jsou s ní v souladu, a tudíž mohou být nezajímavé. Příklady asociačních pravidel, které jsou v souladu s doménovou znalostí *pivo* ↑↑ *BMI*, jsou pravidla Pivo(málo) $\Rightarrow_{p,Base}$ BMI(nízké), Pivo(hodně) $\Rightarrow_{p,Base}$ BMI(vysoké), popř. Pivo(velmi málo) $\Rightarrow_{p,Base}$ BMI(velmi nízké) atd. Předpokládáme, že Pivo je kategoriální atribut, který vznikne vhodnou kategorizací obecného atributu *pivo*, a že Pivo(málo) je booleovský atribut automaticky vytvořený procedurou 4ft-Miner na základě vhodného zadání. Podobné předpoklady uvažujeme pro Pivo(hodně), BMI(nízké) atd. Parametry p, *Base* 4ft-kvantifikátoru $\Rightarrow_{p,Base}$ mohou být zvoleny v závislosti na analyzovaných datech, např. jako $p = 0{,}9$ a $Base = 100$.

Pro doménovou znalost *pivo* ↑↑ *BMI* lze tedy vytvořit množinu $\mathscr{C}_{\Rightarrow}$(*pivo* ↑↑ *BMI*) jednoduchých asociačních pravidel Pivo(α) $\Rightarrow_{p,Base}$ BMI(β), kterou lze chápat jako (možné) atomické důsledky doménové znalosti pivo ↑↑ BMI. Ty lze automaticky odstranit z výstupu procedury 4ft-Miner. Automaticky ale lze odstranit i další asociační pravidla $\varphi \approx \psi$, která jsou logickým důsledkem některého atomického důsledku Pivo(α) $\Rightarrow_{p,Base}$ BMI(β). K tomu lze použít větu o korektnosti dedukčních pravidel zmíněnou v odstavci 11.4.3. Tato věta umožňuje pro každé další asociační pravidlo $\varphi \approx \psi$ zjistit, zda je logickým důsledkem některého atomického důsledku Pivo(α) $\Rightarrow_{p,Base}$ BMI(β). Poznamenejme, že při automatickém zpracování výsledků procedury 4ft-Miner je však třeba opatrnosti. Nízké BMI nemusí být vždy důsledkem nízké konzumace piva, může být i důsledkem přílišného sportovního tréninku.

Existuje více závislostí mezi dvojicemi atributů (Rauch, Šimůnek, 2007), (Rauch, Šimůnek, 2009b), pro které lze sestrojit množiny možných atomických důsledků (Rauch, 2009a), (Rauch, 2009b) a následně využít věty o korektnosti dedukčních pravidel k odstranění asociačních pravidel logicky plynoucích z těchto atomických důsledků. Obdobné věty o korektnosti lze očekávat i pro typy vztahů, se kterými pracují další GUHA procedury. Pro SD4ft pravidlo jsou některé takové věty dokázány v diplomové práci (Kodym, 2007). Naznačené možnosti práce s doménovou znalostí jsou jedním z výzkumných témat, souvisí však s výzkumem observačních kalkulů, jak uvidíme v dalším textu.

4. Ukázali jsme možnosti využití jednoho typu doménových znalostí, vztah *pivo* ↑↑ *BMI* je příkladem takové znalosti. Je však k dispozici **více typů doménových znalostí**, které lze různým způsobem využít. Získávání těchto znalostí, jejich formalizace, uchování a využití je dalším výzkumným tématem, ve kterém jsou již k dispozici významné výsledky (Rauch, Šimůnek, 2009b), (Kliegr a kol., 2009), (Rauch, Šimůnek, 2011). Mezi nejdůležitější typy uchovávaných znalostí doménových znalostí patří skupiny atributů (příklad viz odst. 11.2.2), důležité jsou i významné hodnoty jednotlivých atributů (například hodnoty $BMI \leq 18{,}5$ znamenají podvýživu, viz odst. 11.2.2.3). Dále mezi důležité doménové znalosti patří doporučené způsoby vytváření kategorií jako intervalů na atributech s reálnými hodnotami, popřípadě minimální nebo maximální rozumná délka takových intervalů. Výzkum se týká zejména dalších užitečných forem doménových znalostí, které lze formalizovat a využít při aplikaci GUHA procedur.

5. Kromě naznačeného odfiltrování důsledků doménových znalostí lze doménové znalosti využít i pro **formulaci zajímavých analytických otázek**. Příkladem jsou analytické otázky vycházející ze skupin atributů řešené v odstavci 11.2.2 a oddílu 11.3. Další otázky lze formulovat na základě doménových znalostí týkajících se vztahů dvou atributů, příkladem je zmíněný vztah *pivo* ↑↑ *BMI*. Lze ho využít například k formulaci otázky: „*Platí v dané matici dat výjimky ze vztahu pivo* ↑↑ *BMI?*". Za zajímavou výjimku můžeme považovat například vztah *pivo* $\approx^{KL}$ *BMI* / χ, kde χ je booleovský atribut, který vymezuje množinu pacientů, pro něž platí výjimka, a $\approx^{KL}$ je v tomto případě KL-kvantifikátor, který říká, že když roste hodnota atributu *pivo*, tak hodnota atributu *BMI* klesá. K nalezení takových výjimek je vhodná procedura KL-Miner.

6. Indikátorem výjimky ze vztahu *pivo* ↑↑ *BMI* platící pro pacienty splňující booleovský atribut φ může ale být i dostatečně silné asociační pravidlo $\varphi \wedge$ Pivo(málo) $\Rightarrow_{p,Base}$ BMI(vysoké), viz též již uvedené úvahy. Poznamenejme, že pro formulaci analytických otázek o výjimkách lze stejně jako doménovou znalost *pivo* ↑↑ *BMI* použít i výsledek aplikace GUHA procedury KL-Miner, jenž nám říká, že když roste denní spotřeba piva pacienta, tak roste *BMI* pacienta. V práci (Rauch, Šimůnek, 2008) je popsán návrh takového **systému analytických otázek**. Dopracování tohoto návrhu a implementace softwaru potřebného pro udržování a rozvoj takového systému analytických otázek je další poměrně rozsáhlé výzkumné téma.

7. Doménové znalosti spolu se zkušenostmi s používáním jednotlivých procedur můžeme využít i k **automatizaci práce s jednotlivými procedurami**. Lze automatizovat přípravu dat spočívající ve vytvoření kategorií – intervalů hodnot – pro atributy s reálnými hodnotami i počáteční nastavení vstupů jednotlivých procedur. Toto počáteční nastavení zahrnuje zadání koeficientů jednotlivých atributů, minimální a maximální délky dílčích cedentů, hodnoty parametrů kvantifikátorů atd. Lze automatizovat také odfiltrování důsledků zadaných prvků doménových znalostí a důsledků výsledků jiných procedur. Výzvou je i automatické formulování systému zajímavých analytických otázek a jejich následné automatické řešení pomocí vhodného zřetězení běhu několika procedur. S těmito možnostmi souvisí projekt **EverMiner** formulovaný v příspěvku (Rauch, Šimůnek, 2005b). Automatizace práce s jednotlivými procedurami a projekt EverMiner jsou také samostatným výzkumným tématem.

Je známo, že počet vztahů, které je třeba verifikovat, roste exponenciálně s maximální délkou cedentů, v některých případech i s maximální délkou koeficientů. To znamená, že pro mnoho úloh nelze získat řešení v rozumném čase. Zkušenosti s procedurami SD4f-Miner a Ac4ft-Miner ukázaly, že řadu praktických úloh lze řešit na běžném PC v hodinách, popř. v desítkách hodin. Pro takové úlohy má to, že zrychlíme řešení o řád, dobrý praktický smysl. To vedlo k experimentům s využitím gridu. Ukázalo se, že takto lze dosáhnout významného zrychlení (Šimůnek, Rauch, 2010), což může být velmi důležité pro projekt EverMiner. Využití **gridových technologií** je tedy jedním výzkumných témat souvisejících s rozvojem metody GUHA a systému LISp-Miner.

8. V souvislosti s filtrováním důsledků některých doménových znalostí bylo naznačeno využití dedukčních pravidel v observačních kalkulech. Dedukční pravidla jsou známá jen pro některá asociační pravidla a pro některá SD4ft pravidla. Pro rozšíření možností práce s doménovými znalostmi je tedy zapotřebí poznat dedukční pravidla i pro typy vztahů, se kterými pracují další GUHA procedury. Odpovídající logické kal-

kuly je však nejpre třeba přesně definovat. Takovéto kalkuly je pak možné dále využít i pro poznání závislostí mezi jednotlivými GUHA procedurami. To znamená, že dalším **důležitým výzkumným tématem jsou observační kalkuly**. V souvislosti se záměrem využít metodu GUHA pro multirelační data mining je třeba zkoumat i vícesortové observační kalkuly. Tyto kalkuly byly definovány a studovány v práci (Rauch, 1986) jako zobecnění observačních kalkulů definovaných a studovaných autory knihy (Hájek, Havránek, 1978). Přehled výsledků v tomto směru je v článku (Rauch, 2006).

Metoda GUHA byla dlouhou dobu vyvíjena bez vztahu k databázím. Myšlenka využít metodu GUHA pro analýzu dat uchovávaných v databázích vznikla v druhé polovině sedmdesátých let, viz např. (Rauch, 1979). Podrobně byla tato idea rozpracována v příspěvku (Rauch, 1986). Podstata aplikace metody GUHA v databázích spočívala ve vytvoření GUHA procedury, jejímž vstupem by bylo několik provázaných matic dat a ne pouze jedna matice dat, jako tomu bylo u ostatních GUHA procedur. Několik provázaných matic dat odpovídalo části databáze. Vztahy v datech, se kterými takové procedury pracovaly, vypovídaly o několika provázaných maticích dat. V dnešní terminologii se tento přístup nazývá multirelační data mining (Džeroski, Lavrač, 2001). V práci (Rauch, 1986) jsou definovány dvě multirelační GUHA procedury. Jedna z nich byla implementována a testována v rámci systému LISp-Miner. Další výzkum relačních GUHA procedur probíhal v rámci studentských prací na MFF UK, hlavním výsledkem je implementace relační GUHA procedury v rámci systému Ferda Data Miner (Kuzmin, 2007). Ukazuje se však, že je obtížné využít potenciál multirelační GUHA procedury, je třeba věnovat velkou pozornost podrobné prezentaci a interpretaci nalezených vztahů a využít i doménových znalostí. Přesto však zůstávají **GUHA procedury pro multirelační data mining** důležitým výzkumným tématem.

9. Jedním z obecných výzkumných směrů DZD je **aplikace fuzzy přístupů**. V souvislosti s metodou GUHA byla získána řada výsledků, viz např. (Holeňa, 1996), (Holeňa, 1998), (Holeňa, 2004), (Ivánek, 2004), (Ivánek, 2005), (Ivánek, 2006), přehled dalších je v článku (Hájek a kol., 2010). Příspěvkem k aplikacím fuzzy přístupu k metodě GUHA je také disertační práce (Ralbovský, 2009), kde je k dispozici i přehled ostatních přístupů k fuzzifikaci asociačních pravidel. Práce (Ralbovský 2009) zahrnuje i implementaci fuzzy přístupu k proceduře ASSOC. Aplikace fuzzy přístupů do metody GUHA je však i nadále aktuálním výzkumným tématem.

11.5.2 Projekt SEWEBAR

Jedna z výzev prezentovaných v příspěvku (Qiang, Xindong, 2006) se týká prezentace výsledků DZD uživateli. Jde o to, aby uživatel dostal výsledek spolu se souvislostmi, kterých se výsledek týká, ne pouhý seznam vztahů vyprodukovaných analytickou procedurou. K tomuto cíli směřuje výzkumný projekt SEWEBAR. Jeho podstatou je prezentace výsledků získaných pomocí GUHA procedur uživateli ve formě analytických zpráv dávajících odpovědi na zajímavé analytické otázky. Analytickou zprávou rozumíme vhodně strukturovaný textový dokument poskytující odpověď na zadanou analytickou otázku ve všech potřebných souvislostech.

Předpokládá se, že analytické zprávy budou prezentovány na internetu. Jádrem analytických zpráv budou výsledky aplikací jedné nebo více GUHA procedur z daných dat, výsledky budou doplněny informacemi o řešené analytické otázce, využitých doméno-

vých znalostech i o způsobu, jakým byly použité GUHA procedury aplikovány, včetně vhodné argumentace, proč bylo postupováno uvedeným způsobem a jak uvedené výsledky souvisí s použitými doménovými znalostmi. Budou využívány výsledky dosažené v souvislosti s naznačenými výzkumnými tématy. Důraz však stále bude kladen na srozumitelnost a čtivost analytické zprávy pro uživatele.

Budou využity formalizované doménové znalosti, formalizované postupy pro tvorbu zajímavých analytických otázek i částečně nebo úplně automatizované postupy pro aplikace GUHA procedur. Vazba mezi doménovými znalostmi bude zajištěna například naznačeným způsobem pomocí atomických důsledků a dedukčních pravidel v observačních kalkulech. Výsledkem by tedy měla být uživateli srozumitelná ale také dostatečně formalizovaná analytická zpráva na internetu.

To otevírá možnost, že vhodně zkonstruované automaty budou schopny porozumět zároveň několika takovým analytickým zprávám a vytvořit z nich další analytickou zprávu dávající odpověď na vhodně zformulovanou analytickou otázku. Předpokládáme, že bude možné syntetizovat analytickou zprávu i na základě analytických zpráv týkajících se dvou nebo více datových souborů. Příkladem může být automatická syntéza shrnující analytické zprávy na základě analytických zpráv o kardiologických pacientech v různých nemocnicích. Předpokládáme přitom, že zprávy z jednotlivých nemocnic jsou vystaveny na internetu a splňují nutné požadavky na anonymitu dat. Tento přístup tvorby analytických zpráv odpovídá vizi sémantického webu. Odtud pochází i název projektu SEWEBAR – SEmantic WEB and Analytical Reports. Idea projektu je prezentována v článcích (Rauch, 2007), (Rauch, Šimůnek, 2007), jsou již k dispozici další publikace o jeho různých aspektech a o dílčích výsledcích (Rauch, Šimůnek, 2009b), (Kliegr a kol., 2009).

Poznamenejme, že s projektem SEWEBAR jsou spojeny i úvahy o modifikaci odpovědí na otázky (L0) až (L5) uvedené v práci (Hájek, Havránek, 1978), viz též odst. 11.4.1. Tyto otázky i odpovědi na ně vycházejí z předpokladu podstatného využití matematické statistiky. Racionální induktivní inferenční pravidla umožňující přejít od observačních k teoretickým tvrzením vycházejí ze statistických přístupů. Problém je však v tom, že data v databázích často nesplňují požadavky potřebné ke korektnímu použití statistických přístupů. Navíc, naším cílem nejsou jednotlivá observační tvrzení, která jsou ve vztahu 1 : 1 s odpovídajícími teoretickými tvrzeními, ale analytické zprávy v uvedeném smyslu.

Takové zprávy tedy musí upozorňovat na vlastnosti analyzovaných dat a mimo jiné i na možnost nebo nemožnost korektního použití statistických přístupů k analýze. Může se tedy často stát, že zpráva bude pouze poukazovat na možné závěry s tím, že ale není zajištěna korektnost indukce a že je zapotřebí provést další pozorování. Závěry analýz mohou také vycházet z naznačeného přístupu k odfiltrování nezajímavých důsledků nějaké doménové znalosti. Tento přístup je založen na využití množiny atomických důsledků nějakého prvku doménové znalosti. Příkladem takové množiny je množina jednoduchých asociačních pravidel $\mathscr{C}_{\Rightarrow}(pivo \uparrow\uparrow BMI)$, které lze chápat jako důsledky doménové znalosti *pivo* $\uparrow\uparrow$ *BMI*, viz odst. 11.5.1. Pokud se ve výsledcích podaří nalézt dostatečně mnoho asociačních pravidel, které mohou být důsledkem prozatím neznámé doménové znalosti *pivo* $\uparrow\uparrow$ *cholesterol*, je třeba tento fakt ve zprávě uvést.

Z úvah o analytických zprávách a jejich automatické formulaci plyne, že je vhodné uvažovat o jiném přístupu k odpovědím na otázky (L0) až (L5). Tento přístup bude vycházet z cíle vytvořit formální jazyk pro prezentaci analytických zpráv. Jazyk bude muset umožňovat vyjádřit naznačené vztahy a fakta tak, aby byla srozumitelná pro uživatele, který není odborníkem v DZD. Potřebná vysvětlení a zdůvodnění však mohou být uvedeny v samostatných kapitolách analytické zprávy určené expertovi na DZD.

Lze předpokládat, že výzkum naznačených výzkumných témat i projekt SEWEBAR přispějí k rozvoji oboru DZD.

Literatura

Agrawal R., Imielinski T., Swami A.: Mining Association Rules between Sets of Items. in Large Databases. In: *Proceedings of the 1993 ACM SIGMOD Conference Washington DC*, USA, May 1993, s. 207–216.

Agrawal R., Mannila H., Srikant R., Toivonen H., Verkamo A. I.: Fast Discovery of Association Rules. In: Fayyad U. a kol. (eds): *Advances in Knowledge Discovery and Data Mining.* AAI Press/MIT Press, 1996, s. 307–328.

Berka P., Rauch J., Tomečková M.: Data Mining in Atherosclerosis Risk Factor Data. In: Berka P., Rauch J., Zighed D. A.: *Data Mining and Medical Knowledge Management: Cases and Applications.* IGI Global, Hershey 2009, s. 376–397.

Berka P.: *Dobývání znalostí z databází.* Academia, Praha 2003.

Džeroski S., Lavrač N. (eds): *Relational Data Mining.* Springer-Verlag, Berlín 2001.

Fayyad U., Piatetstki-Shapiro G., Smyth P., Uthurusamy R. (eds): *Advances in Knowledge Discovery and Data Mining.* AAI Press/MIT Press, 1996.

Hájek P., Havel I., Chytil M.: The GUHA method of automatic hypotheses determination, *Computing* 1, 1966, s. 293–308.

Hájek P. (ed.): *International Journal of man-Machine* Studies 1, vol. 10 (special issue on GUHA), 1978.

Hájek P. (ed.): *International Journal of man-Machine Studies* 3, vol. 15, (second special issue on GUHA), 1981.

Hájek P., Havel I., Havránek T., Chytil M., Rauch J., Renc Z.: *Metoda GUHA.* Dům techniky ČVTS České Budějovice, 1976 (1. vyd.), 1977 (2. vyd.).

Hájek P., Havránek T., Chytil M.: *Metoda GUHA – automatická tvorba hypotéz.* Academia, Praha 1983.

Hájek P., Havránek T.: Mechanising *Hypothesis Formation – Mathematical Foundations for a General Theory.* Springer-Verlag, Berlín, Heidelberg, New York 1978.

Hájek P., Holeňa M., Rauch J.: The GUHA method and its meaning for data mining, *Journal of Computer and System Science* 1, **76**, 2010, s. 34–48.

Hájek P.: Metoda GUHA v minulém století a dnes. In: Snášel V. (ed.): *Znalosti 2004.* VŠB TU Ostrava, Ostrava, 2004.

Holeňa M.: Exploratory data processing using a fuzzy generalization of the Guha approach. In: Baldwin J. F. (ed.): *Fuzzy Logic.* John Wiley and Sons, New York 1996, s. 213–229.

Holeňa M.: Fuzzy hypotheses for Guha implications, *Fuzzy Sets and Systems* 98, 1998, s. 101–125.

Holeňa M.: Fuzzy hypotheses testing in the framework of fuzzy logic, *Fuzzy Sets and Systems* 145, 2004, s. 229–252.

Chrz M.: *Transparent deduction rules for GUHA procedures*. diplomová práce, MFF UK, 2007.

Iványek J.: Using Fuzzy Logic Operators for Construction of Data Mining quantifiers, *Neural Network World* 5, **14**, 2004, s. 403–410.

Iványek J.: Construction of implicational quantifiers from fuzzy implications, *Fuzzy Sets and System* 2, **151**, 2005, s. 381–391.

Iványek J.: Combining Implicational Quantifiers for Equivalence Ones by Fuzzy Connectives, *International Journal of Intelligent Systems* 3, **21**, 2006, s. 325–334.

Kliegr T. a kol.: Semantic Analytical Reports: A Framework for Post-processing data Mining Results. In: *Foundations of Intelligent Systems*. Springer-Verlag, Berlín 2009, s. 88–98.

Kodym J.: *Třídy SD4ft-pravidel*. diplomová práce, MFF UK, 2007.

Kuzmin A.: *Relační GUHA procedury*. diplomová práce, MFF UK, 2007.

Lín V., Dolejší P., Rauch J., Šimůnek M.: The KL-Miner Procedure for Datamining, *Neural Network World* 5, 2004, s. 411–420.

Qiang Y., Xindong W.: 10 Challenging Problems in Data Mining Research, *International Journal of Information Technology & Decision Making* 4, vol. 5, 2006, s. 597–604.

Ralbovský M., Kuchař, T.: Using Disjunctions in Association Mining. In: *Advances in Data Mining*. Springer-Verlag, Berlín, 2007, s. 339–351.

Ralbovský M.: *Fuzzy GUHA – Theory and Implementation of Fuzzy Association Rules.* disertační práce, Fakulta informatiky a statistiky Vysoké školy ekonomické v Praze, Praha 2009.

Ras Z., Wieczorkowska A.: Action-Rules: How to Increase Profit of a Company. In: Zighed D A, Komorowski J, Zytkow J: *Principles of Data Mining and Knowledge Discovery*. LNCS/LNAI, No. 1910, Springer-Verlag, 2000, s. 587–592.

Rauch J., Šimůnek M., Lín, V.: Mining for Patterns Based on Contingency Tables by KL-Miner – First Experience. In: Lin T. Y. a kol. (eds): *Foundations and Novel Approaches in Data Mining.* Springer-Verlag, Berlín 2005, s. 155–167.

Rauch J., Šimůnek M.: An Alternative Approach to Mining Association Rules. In: Lin T. Y. a kol. (eds). *Foundations of Data Mining and Knowledge Discovery*. Springer, Berlín 2005a, s. 211–231.

Rauch J., Šimůnek M.: GUHA Method and Granular Computing. In: Hu X. a kol. (eds): *Proceedings of IEEE conference Granular Computing*, Piscataway 2005b, s. 630–635.

Rauch J., Šimůnek M.: Semantic Web Presentation of Analytical Reports from Data Mining – Preliminary Considerations. In: WEB INTELLIGENCE, *IEEE Computer Society*, Los Alamitos 2007, s. 3–7.

Rauch J., Šimůnek M.: LAREDAM – Considerations on System of Local Analytical Reports from Data Mining. In: *Foundations of Intelligent Systems.* Springer-Verlag, Berlín 2008, s. 143–149.

Rauch J., Šimůnek M.: Action Rules and the GUHA Method: Preliminary Considerations and Results. In: *Foundations of Intelligent Systems*. Springer-Verlag, Praha, Berlín 2009a, s. 76–87.

Rauch J., Šimůnek M.: Dealing with Background Knowledge in the SEWEBAR Project. In: Berendt B. a kol. (eds): *Knowledge Discovery Enhanced with Semantic and Social Information.* Springer-Verlag, Berlín 2009b, s. 89–106.

Rauch J., Šimůnek M.: Applying Domain Knowledge in Association Rules Mining Process – First Experience. In: *ISMIS 2011 Foundations of Intelligent Systems.* Springer, Londýn 2011, s. 113–122.

Rauch J., Tomečková M.: System of Analytical Questions and Reports an Mining in Health Data – a Case Study. In: *mccsis 2007* [CD-ROM], IADIS, Lisabon 2007, s. 176–181.

Rauch J.: Some Remarks on Computer Realisations of GUHA Procedures, *International Journal of Man-Machine Studies* **10**, 1978, s. 23–28.

Rauch J.: Dotazovací jazyky a automatická tvorba hypotéz. In *Sborník SOFSEM'79*, VVS Bratislava, 1979, s. 388–389.

Rauch J.: *Logické základy automatické tvorby hypotéz z databází.* disertační práce, Matematický ústav ČSAV, Praha 1986, 133 s.

Rauch J.: *Příspěvek k logickým základům KDD.* habilitační práce, Fakulta informatiky a statistiky Vysoké školy ekonomické v Praze, 1998a.

Rauch J.: Classes of Four-Fold Table Quantifiers. In: Zytkow J., Quafafou M. (eds): *Principles of Data Mining and Knowledge Discovery.* Nantes 1998b, s. 203–211.

Rauch J.: Logic of Association Rules, *Applied Intelligence* 22, 2005, s. 9–28.

Rauch J.: Many Sorted Observational Calculi for Multi-Relational Data Mining. In: Data Mining – Workshops, *IEEE Computer Society*, 2006, s. 417–422.

Rauch J.: Project SEWEBAR – Considerations on Semantic Web and Data Mining. In: *IICAI-07* [CD-ROM]. A&M University, Tallahassee, Florida, 2007, s. 1763–1782.

Rauch J.: Classes of Association Rules: an Overview. In: Lin T. Y. a kol. (eds): *Data Mining: Foundations and Practice.* Springer, Berlín 2008, s. 315–337.

Rauch J.: Considerations on Logical Calculi for Dealing with Knowledge in Data Mining. In: Ras Z. W., Dardzinska A.: *Advances in Data Management.* Springer-Verlag, Berlín 2009a, s. 177–201 (Studies in Computational Intelligence 223/2009).

Rauch J.: The GUHA method and observational calculi: tools for data mining. In: Cintula P., Haniková Z., Švejdar V. (eds): *Witnessed Years – Essays in Honour of Petr Hájek.* College Publications, Londýn 2009b, s. 103–127.

Rauch J.: Logical Aspects of the Measures of Interestingness of Association Rules. In: Koronacki J. a kol. (eds): *Recent Advances in Machine Learning (Dedicated to the memory of Ryszard S. Michalski).* Springer, Berlín, Heidelberg 2010, s. 175–203.

Rauch J.: *Observational Calculi and Association Rules.* Springer, Heidelberg 2013.

Šimůnek M.: *Systém LISp-Miner, akademický systém pro dobývání znalostí z databází, historie vývoje a popis ovládání.* 1. vyd., Oeconomica, Praha 2010.

Šimůnek M., Rauch J.: Rozšíření systému LISp-Miner o distribuované řešení úloh prostřednictvím PC-Gridu. In: *Znalosti 2010.* Oeconomica, Praha 2010, s. 267–270.

Šimůnek M.: Academic KDD Project LISp-Miner. In: Aberaham M. A., Franke K., Koppen, K. (eds): *Advances in Soft Computing – Intelligent Systems Desing and Applications.* Springer-Verlag, Heidelberg 2003, s. 263–272.

Xindong W. a kol.: Top 10 Algorithms in Data Mining, *Knowledge and Information Systems* 1, **14**, 2007, s. 1–37.

12 Taxonomie výpočetních modelů neuronových sítí: od subregulárních jazyků k superturingovským výpočtům

12.1 Úvod

12.1.1 Výpočetní teorie neuronových sítí

Takzvané (*umělé*) *neuronové sítě* představují velmi zjednodušené matematické modely nervových systémů živých organismů. Ve snaze pochopit a modelovat biologické a kognitivní funkce lidského mozku se tyto modely stále zdokonalují, aby věrněji odrážely současné neurofyziologické poznatky (Gerstner, Kistler, 2002). Na druhé straně tohoto úsilí stojí inženýři, kteří modely neuronových sítí modifikují, aby je mohli využít pro řešení praktických úloh z umělé inteligence (Haykin, 1999). Je zajímavé, že již konstruktéři prvních počítačů chtěli napodobit lidskou inteligenci, a proto se při své práci inspirovali funkcí mozku (von Neumann, 1951), (von Neumann, 1956). Například první neuropočítač (Minsky, 1954) byl zkonstruován v roce 1951 nedlouho poté, co byl vyvinut první konvenční počítač von neumannovské architektury. Neuronové sítě se dále profilovaly jako samostatný obor informatiky. Hlavní předností neuronových sítí je jejich schopnost učit se z dat, tedy zobecňovat požadovanou funkci z příkladů. Neuronové sítě jako netradiční výpočetní prostředky inspirované přírodou byly úspěšně aplikovány v oblastech, kde je návrh exaktních algoritmů problematický, např. při rozpoznávání obrazců, řízení, predikci, rozhodování, analýze a transformaci signálů, detekci chyb, v expertních systémech apod. Současné komerční programy pro analýzu a zpracování dat (např. *Matlab*, *Statistica*) obvykle obsahují moduly pro práci s neuronovými sítěmi, které představují velmi účinný nástroj v mnoha praktických aplikacích strojového učení.

Úvod do oboru neuronových sítí lze najít v 9. kapitole prvního dílu této publikace (Hořejš, 1993), zatímco s pokročilejšími technikami se lze seznámit v 7. a 9. kapitole čtvrtého dílu (Jiřina, 2003), (Grim, 2003). Cílem této kapitoly je prezentovat taxonomii formálních výpočetních modelů neuronových sítí v rámci jednotného formalismu spolu s přehledem jejich výpočetních vlastností. Vycházíme přitom z přehledové práce (Šíma, Orponen, 2003c). Výpočetní potenciál a limity konvenčních počítačů jsou studovány například pomocí Turingových strojů, u kterých je případně omezena doba výpočtu či

velikost paměti; srovnej se 7. kapitolou třetího dílu této publikace (Demlová, Štěpánková, 2001). Analogicky bylo v posledních 20 letech dosaženo mnoho fundamentálních výsledků týkajících se schopnosti neuronových sítí implementovat univerzální výpočty (Anthony, 2001), (Floréen, Orponen, 1994), (Orponen, 1994), (Parberry, 1994), (Roychowdhury a kol., 1994), (Siegelmann, 1999a), (Siu a kol., 1995), (Šíma, Neruda, 1996), (Wiedermann, 1994). Výpočetní síla neuronových sítí je obvykle studována tak, že různé modely sítí porovnáváme nejen mezi sebou, ale především s tradičními výpočetními modely a popisnými prostředky, jako jsou konečné automaty, regulární výrazy, gramatiky, Turingovy stroje, booleovské obvody apod. Cílem tohoto přístupu je zjistit, co lze v principu nebo efektivně spočítat pomocí jednotlivých modelů neuronových sítí, resp. jak implementovat požadovanou funkci optimálním způsobem. Hlavním technickým nástrojem tohoto výzkumu je výpočetní teorie složitosti (Balcázar a kol., 1995), (Papadimitriou, 1994), popř. formální teorie jazyků (Hopcroft, Ullman, 1979), (Chytil, 1984), proto v příloze této kapitoly uvádíme technické definice některých složitostních tříd, i když předpokládáme, že čtenář je seznámen se základními abstraktními výpočetními modely, jakými jsou konečné automaty, Turingovy stroje apod.

Hluboký teoretický vhled a porozumění principům a možnostem neurových výpočtů je předpokladem pro volbu adekvátního modelu a parametrů neuronové sítě při řešení daného problému nebo pro návrh efektivnějších učících algoritmů. Vzhledem k tomu, že modely neuronových sítí jsou inspirovány neurofyziologickými poznatky, teoretické výsledky mohou kromě praktického využití prohloubit naše chápání výpočetních aspektů mentálních procesů. Navíc neuronové sítě obohacují tradiční repertoár formálních výpočetních modelů, které jsou obvykle parametrizovány časem a prostorem (časová a paměťová složitost), o nové zdroje efektivního počítání, jimiž jsou například *analogové stavy* (Siegelmann, 1999a), *spojitý čas* (Orponen, 1997c), *energie* (Šíma, 2003), (Uchizawa a kol., 2006) apod. Protože například tradiční Turingův stroj tyto parametry nezná a příslušné modely neuronových sítí mnohdy formálně implementují superturingovské výpočty, neuronové sítě v tomto případě aspirují na to, aby se staly referenčními modely při studiu těchto alternativních výpočetních zdrojů.

Výpočetní aspekty neuronových sítí lze studovat ve třech hlavních směrech: efektivním vytváření neboli adaptaci reprezentace neuronové sítě (*složitosti učení a generalizací*), její paměťové náročnosti (*deskriptivní složitosti*) a efektivní odezvě neuronové sítě na její vstup (*výpočetní síle*). V této kapitole se zaměříme pouze na deskriptivní složitost a výpočetní sílu neuronových sítí, zatímco úplně opomineme důležité výsledky o složitosti učení, které by zasluhovaly samostatný přehled (Anthony, Bartlett, 1999), (Roychowdhury a kol., 1994), (Šíma, 2002), (Vidyasagar, 1997). Dále se omezíme jen na *digitální* výpočty, jejichž vstupy a výstupy jsou ve své podstatě diskrétní veličiny, i když jsou často kódovány pomocí analogových stavů neuronů a vlastní výpočet neuronové sítě může pracovat s reálnými čísly. Naopak výpočetní schopnosti a efektivita modelů s *analogovými* vstupy jsou studovány v teorii aproximace funkcí pomocí neuronových sítí, kterou se zabývá 8. kapitola čtvrtého dílu této publikace (Kůrková, 2003).

12.1.2 Taxonomie formálních výpočetních modelů neuronových sítí

Nejprve si připomeneme obecnou definici modelu neuronové sítě, která ve své neurčitosti pokrývá většinu publikovaných a používaných modelů a kterou budeme dále kon-

kretizovat při specifikaci jednotlivých modelů. *Neuronová síť* se skládá z s jednoduchých výpočetních *jednotek*, neboli *neuronů*, z množiny $V = \{1,..., s\}$, kde počet neuronů $s = |V|$ se nazývá *velikost sítě*. Některé z těchto jednotek mohou sloužit jako externí vstupy nebo výstupy, proto předpokládáme, že síť má n *vstupních* a m *výstupních* neuronů. Tyto jednotky jsou hustě propojeny do orientovaného grafu, který reprezentuje *architekturu* (neboli *topologii*) sítě, v níž je každá hrana (i, j) vedoucí z neuronu i do neuronu j ohodnocena reálnou (*synaptickou*) *váhou* $w(i, j) = w_{ij} \in \Re$. Absence spoje v architektuře odpovídá nulové váze mezi příslušnými neurony a naopak.

Výpočetní dynamika neuronové sítě určuje vývoj reálného *stavu* (*výstupu*) $y_j^{(t)} \in \Re$ pro každý neuron $j \in V$ jako funkci času $t \geq 0$. Souhrnně je tak definován *stav sítě* $\mathbf{y}^{(t)} = (y_1^{(t)},..., y_s^{(t)}) \in \Re^s$ v každém časovém okamžiku $t \geq 0$. Na začátku výpočtu se síť nachází v *počátečním stavu* $\mathbf{y}^{(0)}$, který může kódovat externí vstup pomocí počátečních stavů vstupních neuronů. Stav sítě je typicky aktualizován tak, že podmnožina neuronů načte své vstupy od incidenčních neuronů přes příslušné zvážené spoje a transformuje je na své aktuální výstupy. Globální výstup sítě je odečten ze stavů výstupních neuronů na konci nebo někdy i v průběhu výpočtu. Jak již bylo řečeno, budeme se zabývat digitálními výpočty, a proto předpokládáme *binární* externí vstupní a výstupní hodnoty.

Výpočetní modely neuronových sítí lze klasifikovat podle typu architektury na dopředné (acyklické) nebo rekurentní (cyklické), podle časové dynamiky na sítě diskrétní nebo spojité, podle typu výpočtu pak na deterministické nebo pravděpodobnostní, podle

Obr. 12.1 Taxonomie modelů perceptronových sítí.

domény stavů neuronů na sítě binární nebo analogové, podle omezení kladených na váhové parametry na symetrické nebo asymetrické, podle typu vstupního protokolu na sítě konečné nebo nekonečné posloupnosti sítí atd. Kombinací těchto omezení dostáváme bohatou taxonomii modelů neuronových sítí, které mají různou výpočetní sílu. V našem přehledu se zaměříme především na tradiční perceptronové sítě a v závěru se jen zmíníme o modelech, které využívají jiných typů jednotek. Kromě jednotek RBF

(Radial Basis Function) a WTA (Winner-Take-All) půjde o biologicky více plausibilní pulsní (spiking) neurony, které používají časového kódování informace a jsou v poslední době předmětem intenzivního výzkumu. Struktura této kapitoly částečně sleduje taxonomii modelů perceptronových sítí, která je znázorněna na obr. 12.1, kde jsou také vyznačeny odkazy na příslušné oddíly či odstavce.

12.1.3 Perceptronové sítě

Nejpopulárnějším modelem neuronových sítí jsou perceptronové sítě. Jak je zvykem, termín *perceptron* je zde chápán obecněji než v původní Rosenblattově práci (Rosenblatz, 1958) a pokrývá i typy jednotek, které z něj byly později odvozeny, např. sigmoidální jednotky. Zde se zaměříme zejména na výpočetní aspekty *diskrétní* perceptronové sítě, která aktualizuje svůj stav jen v diskrétních časových krocích $t = 1, 2,...$

V čase $t \geq 0$ se nejprve vyhodnotí *excitace* (*vnitřní potenciál*) každého neuronu $j \in V$ jako zvážený součet jeho vstupů

$$\xi_j^{(t)} = \sum_{i=0}^{s} w_{ji} y_i^{(t)} . \tag{12.1}$$

Uvedená suma navíc obsahuje absolutní člen, tzv. *bias* $w_{j0} \in \Re$, který lze chápat jako váhu přidaného formálního jednotkového vstupu $y_0^{(t)} \equiv 1$. Připomeňme, že některé váhy w_{ji} $(1 \leq i \leq s)$ ve vzorci (12.1) mohou být nulové, což značí absenci spoje z neuronu i do neuronu j. V následujícím časovém kroku $t + 1$ neurony $j \in \alpha_{t+1}$ z vybrané podmnožiny $\alpha_{t+1} \subseteq V$ počítají svůj nový výstup $y_j^{(t+1)}$ pomocí *aktivační* (*přenosové*) *funkce* $\sigma: \Re \to \Re$:

$$y_j^{(t+1)} = \sigma\left(\xi_j^{(t)}\right) \quad \text{pro } j \in \alpha_{t+1}, \tag{12.2}$$

zatímco zbývající jednotky $j \notin \alpha_{t+1}$ své stavy nemění, tj. $y_j^{(t+1)} = y_j^{(t)}$ pro $j \notin \alpha_{t+1}$. Tímto způsobem je určen nový stav sítě $\boldsymbol{y}^{(t+1)}$ v čase $t + 1$.

Perceptronové sítě s *binárními stavy* $y_j \in \{0, 1\}$ (zkráceně *binární sítě*) obvykle používají *Heavisideovu* aktivační funkci

$$\sigma(\xi) = \begin{cases} 1 & \text{pro } \xi \geq 0 \\ 0 & \text{pro } \xi < 0 \end{cases} \tag{12.3}$$

a odpovídající jednotky se také nazývají *prahová hradla*. V případě potřeby je možné binární stavové hodnoty {0, 1} nahradit bipolárními {–1, 1} nebo dokonce obecnějšími diskrétními doménami, což zásadním způsobem nemění velikost váhových parametrů (Parberry, 1994). Váhy binárního perceptronu lze při zachování jeho funkce také upravit tak, aby excitace (12.1) byla nenulová pro všechny vstupy.

Na druhou stranu sítě s *analogovými stavy* (zkráceně *analogové sítě*) aproximují Heavisideovu funkci (12.3) nějakou spojitou sigmoidální aktivační funkcí, např. *saturovanou lineární* funkcí

$$\sigma(\xi)=\begin{cases}1 & \text{pro } \xi \geq 1\\ \xi & \text{pro } 0<\xi<1\\ 0 & \text{pro } \xi \leq 0\end{cases} \tag{12.4}$$

nebo *logistickou* (*standardní*) *sigmoidou*

$$\sigma(\xi)=\frac{1}{1+e^{-\xi}} \tag{12.5}$$

a odpovídající jednotky se také nazývají *sigmoidální hradla*. Tedy výstupy ze sigmoidálních hradel jsou obecně reálnými čísly, např. $y_j \in [0, 1]$. Pro potřeby výpočtu booleovských funkcí jsou analogové stavy y_j výstupních neuronů j interpretovány jako binární výstupy 0, popř. 1, se *separací* $\varepsilon > 0$, jestliže $y_j \leq h - \varepsilon$, popř. $y_j \geq h + \varepsilon$, pro danou pevnou prahovou hodnotu $h \in \Re$.

12.2 Jednotlivý perceptron

Jednotlivý perceptron s n externími binárními vstupy zřejmě počítá booleovskou funkci $f: \{0, 1\}^n \rightarrow \{0, 1\}$ o n proměnných. Je všeobecně známo, že ne všechny booleovské funkce lze implementovat pomocí jednoho perceptronu. Prominentním příkladem je parita (XOR), která nabývá hodnoty 1, právě když je počet vstupních bitů s hodnotou 1 lichý (Minsky, Papert, 1969). Naopak booleovské funkce, které lze počítat jedním perceptronem, se nazývají *lineární prahové funkce* (Muroga, 1971). Tato důležitá třída funkcí obsahuje základní logické funkce AND, OR, NOT a je uzavřená na negaci funkce a negaci vstupních proměnných, tedy negaci libovolné booleovské lineární prahové funkce s možnou negací některých jejích proměnných lze opět počítat pomocí jednoho perceptronu. Počet booleovských lineárních prahových funkcí n proměnných je řádově $2^{\Theta(n^2)}$. Horní odhad $2^{n^2-n\log_2 n+O(n)}$ pochází již od Schläfliho (1901), zatímco přesnější, přiléhavý dolní odhad $2^{n^2-n\log_2 n-O(n)}$ byl dokázán o dost později (Kahn a kol., 1995). To znamená, že jen velmi malý zlomek ze všech 2^{2^n} booleovských funkcí lze počítat pomocí jednoho perceptronu. Navíc je obecně algoritmicky těžké (co-NP-úplné) rozhodnout, zda konkrétní booleovská funkce daná v disjunktivní nebo konjunktivní normální formě je lineární prahová (Hegedüs, Megiddo, 1996).

Také byla zkoumána deskriptivní složitost booleovských lineárních prahových funkcí. Je známo, že každou takovou funkci lze implementovat perceptronem, který má jen *celočíselné* váhy (Minsky, Papert, 1969), a počet bitů, které jsou potřeba (Håstad, 1994) a stačí (Muroga a kol., 1961) na reprezentaci jedné celočíselné váhy perceptronu s n vstupy, je řádově $\Theta(n \log n)$. Tyto odhady mají velký význam pro hardwarovou, popř. softwarovou, implementaci perceptronu.

12.3 Dopředné sítě

Architekturu *dopředné* (*vrstvené, vícevrstvé, acyklické*) sítě tvoří acyklický graf. To znamená, že jednotky v dopředné síti lze vždy jednoznačně seskupit do nejkratší posloupnosti $d + 1$ (po dvou disjunktních) *vrstev* $\alpha_0,\ldots, \alpha_d \subseteq V$ tak, aby neurony v libovolné vrstvě α_t byly napojeny jen na neurony v následujících vrstvách α_u, $u > t$. Obvykle se první – tzv. *vstupní vrstva* α_0, která se skládá z n externích vstupů – nepočítá do počtu vrstev a velikosti sítě. Podobně poslední, tzv. *výstupní vrstva* α_d, je tvořena m výstupními neurony. Skryté jednotky se pak nachází v mezilehlých, tzv. *skrytých vrstvách* $\alpha_1,\ldots, \alpha_{d-1}$. Počet vrstev d vyjma vstupní vrstvy se nazývá *hloubka sítě* (např. dvouvrstvé sítě mají hloubku 2). Výpočet sítě postupuje vrstvu po vrstvě směrem od vstupní vrstvy přes skryté vrstvy až k vrstvě výstupní. Na počátku jsou stavy vstupních neuronů nastaveny na externí vstup. V obecném kroku předpokládejme, že výstupy všech neuronů jsou určeny až do jisté vrstvy $t < d$ včetně. Stavy neuronů v následující vrstvě α_{t+1} jsou vypočteny paralelně podle vzorců (12.1) a (12.2). Nakonec stavy výstupních neuronů z α_d představují výsledek výpočtu. Tímto způsobem je vyhodnocena tzv. *funkce sítě* $\mathbf{f}: \{0, 1\}^n \to \{0, 1\}^m$ v paralelním čase, který se rovná hloubce sítě d.

Popsaný model dopředné neuronové sítě splývá s modelem *booleovských prahových obvodů* (dále jen *obvodů*), jejichž výpočetní jednotky se obvykle nazývají (prahová) hradla. Výpočetní aspekty obvodů jsou předmětem intenzivního výzkumu v teorii složitosti booleovských obvodů (Vollmer, 1999), (Wegener, 1987). Jednotlivý obvod C s n vstupy počítá (vektorovou) booleovskou funkci $\mathbf{f}$, která závisí na pevném počtu proměnných n. Pro potřeby univerzálních výpočtů nad vstupy libovolné délky se používá *nekonečná posloupnost* (*rodina*) obvodů $\{C_n\}$, která obsahuje jeden obvod C_n pro každou délku vstupu $n \geq 0$. V tomto kontextu jsou pak míry složitosti obvodu jako velikost $S(n)$, hloubka $D(n)$, maximální váha $W(n)$ atd. vyjádřeny jako funkce délky vstupu n. Například pro posloupnost obvodů *polynomiální velikosti* je funkce $S(n)$ omezena nějakým polynomem $O(n^c)$ (kde c je konstanta), nebo posloupnost obvodů *konstantní hloubky* splňuje $D(n) \leq c$ apod.

Nekonečné posloupnosti obvodů mohou být podle uvedené definice obecně *neuniformní* v tom smyslu, že neexistuje jejich konečný popis, který by dával do souvislosti obvody pro různé délky vstupu. To znamená, že takové nekonečné posloupnosti obvodů mohou implementovat funkce, které nelze počítat pomocí konečně popsaného algoritmu (např. Turingova stroje), a mají tedy „superturingovské" výpočetní schopnosti, což je triviální důsledek jejich definice. Tomu se lze vyhnout tak, že předpokládáme tzv. *uniformní* posloupnost obvodů, pro kterou existuje konečný algoritmus (pevný Turingův stroj), který pro každou délku vstupu $n \geq 0$ zkonstruuje příslušný obvod C_n z této posloupnosti. Je zřejmé, že uniformní posloupnosti obvodů jsou výpočetně ekvivalentní Turingovým strojům.

12.3.1 Binární dopředné sítě

V tomto odstavci se budeme zabývat výpočetními vlastnostmi binárních dopředných sítí s Heavisideovou aktivační funkcí (12.3).

12.3.1.1 Výpočetní univerzalita

Nejjednodušší případ tohoto modelu, jednotlivý perceptron, jsme již uvažovali v oddílu 12.2, ze kterého je zřejmé, že prahová hradla musí být zapojena do větších obvodů, aby byla schopna implementovat všechny booleovské funkce. Konkrétně pro libovolnou vektorovou booleovskou funkci $\boldsymbol{f} : \{0, 1\}^n \rightarrow \{0, 1\}^m$ lze vždy zkonstruovat prahový obvod hloubky 4, který počítá $\boldsymbol{f}$ s použitím

$$s = \Theta\left(\sqrt{\frac{m2^n}{n - \log m}}\right) \tag{12.6}$$

hradel (Lupanov, 1972). Tato konstrukce je optimální v tom smyslu, že existují funkce, které dokazatelně vyžadují uvedený počet hradel i v případě, že povolíme neomezenou hloubku obvodu (Horne, Hush, 1994). Odpovídající dolní (Nechiporuk, 1964) a horní (Lupanov, 1961) odhad byl nejprve stanoven pro sítě s jedním výstupem a souhlasí se vzorcem (12.6) pro $m = 1$. Navíc Cover (1968) dokázal, že počet spojů, který je v nejhorším případě potřeba v prahovém obvodu počítajícím libovolnou booleovskou funkci, lze vyjádřit jako čtverec formule (12.6). Tyto výsledky znamenají, že (neuniformní) nekonečné posloupnosti prahových obvodů konstantní hloubky s exponenciálním počtem hradel (vzhledem k počtu vstupů) jsou schopny počítat libovolnou vstupně–výstupní funkci v konstantním paralelním čase.

12.3.1.2 Polynomiální váhy

V booleovské složitosti se kladou další omezení na parametry obvodů. Například v prahových obvodech jsou celočíselné váhové parametry obvykle omezeny polynomem $O(n^c)$ (pro nějakou konstantu c) vzhledem k počtu vstupů obvodu n, což odpovídá $O(\log n)$ bitům na reprezentaci jedné váhy. Polynomiální váhy zřejmě snižují výpočetní sílu prahových obvodů, protože již z oddílu 12.2 víme, že perceptron vyžaduje v nejhorším případě exponenciální váhy. Nicméně Goldmann a kol. (1992) dokázali, že neomezené váhy v prahových obvodech lze principiálně vždy (za cenu jen polynomiálního nárůstu velikosti obvodu) nahradit polynomiálními váhami, pokud zvýšíme hloubku obvodu o jednu vrstvu. Goldmann a Karpinski (1998) dále navrhli explicitní konstrukci takového obvodu s polynomiálními váhami, u něhož polynomiální nárůst velikosti nezávisí na hloubce obvodu. Tedy u vícevrstvé perceptronové sítě lze vždy předpokládat polynomiální váhy, pokud navíc připustíme jeden výpočetní krok.

Jiná konvence předpokládá, že hodnoty booleovských proměnných vstupují do obvodu vždy spolu se svými negacemi. Za tohoto předpokladu lze každý prahový obvod upravit tak, aby obsahoval jen *kladné* váhy, zatímco jeho velikost se nejvýše zdvojnásobí a jeho hloubka a velikost vah se nezmění (Hajnal a kol., 1993).

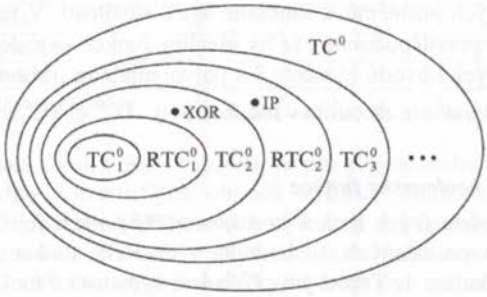
12.3.1.3 Omezený vstupní stupeň hradel

Jiné omezení se týká maximálního *vstupního stupně* hradel v obvodu, tj. maximálního počtu vstupů u jednotlivých neuronů. Obvody s omezeným vstupním stupněm modelují konvenční technologii obvodů, u nichž je počet vstupů hradla omezený. Neomezený vstupní stupeň je naopak typický pro hustě propojené neuronové sítě, které díky tomu

mohou být výpočetně efektivnější. Například každý obvod velikosti s a hloubky d s maximálním vstupním stupněm 2 je možné implementovat pomocí prahového obvodu s neomezeným vstupním stupněm velikosti $O(s^2)$ hradel, jehož hloubka je řádově $O(d/\text{loglog}s)$ (Chandra a kol., 1984). V neuronových sítích je tak možné v rámci polynomiální velikosti dosáhnout zrychlení paralelního výpočtu o faktor $O(\text{loglog}n)$.

12.3.1.4 Polynomiální velikost a konstantní hloubka

Univerzální konstrukci obvodu exponenciální velikosti (12.6) nelze v praxi realizovat pro větší počty vstupů. Na druhou stranu o žádné pro praxi „zajímavé" funkci (např. ze složitostní třídy NP) nebylo zatím dokázáno, že by ke své implementaci potřebovala více než lineární počet hradel obvodu s neomezenou hloubkou, i když intuitivně nejsou funkce vyžadující velké obvody žádnou vzácností (Wegener, 1987). Nicméně mnoho důležitých funkcí je možné počítat pomocí prahových obvodů polynomiální velikosti a konstantní hloubky (viz odstavce 12.3.1.5 a 12.3.1.6), a tedy je lze reálně implementovat. Navíc jsou tyto výsledky ve shodě s neurofyziologickým pozorováním, že komplikované funkce jsou v mozku realizovány s využitím jen několika málo vrstev neuronů.



Obr. 12.2 Možná TC^0-hierarchie se separací malých hloubek.

Třída všech funkcí, které lze počítat pomocí prahových obvodů polynomiální velikosti a hloubky $d \geq 1$ s polynomiálními váhami, se obvykle značí TC^0_d (alternativní značení je LT_d, kde LT_d bez stříšky odkazuje na odpovídající třídu s neomezenými váhami). Složitostní třída $TC^0 = \bigcup_{d \geq 1} TC^0_d$ pak obsahuje všechny funkce, které lze pomocí takových obvodů počítat v konstantním paralelním čase. Zřejmě

$$TC^0_1 \subseteq TC^0_2 \subseteq TC^0_3 \subseteq \cdots \tag{12.7}$$

je tedy možná TC^0-*hierarchie*, která je znázorněna na obrázku 12.2 spolu s odpovídajícími pravděpodobnostními třídami složitosti RTC^0_d, jimiž se budeme zabývat dále v odst. 12.5.1. Například třída TC^0_1, jež obsahuje všechny funkce, které lze počítat jednotlivými perceptrony s polynomiálními váhami, je jistě vlastní podtřídou třídy TC^0_2, tj.

$TC_1^0 \subsetneq TC_2^0$, protože například paritu (XOR) nelze počítat jedním perceptronem, ale lze ji jednoduše implementovat malým obvodem hloubky 2.

Méně známým výsledkem je netriviální separace $TC_2^0 \subsetneq TC_3^0$ (Hajnal a kol., 1993), která má velký význam v praktických aplikacích dopředných neuronových sítí, kde se obvykle rozhoduje o použití dvouvrstvé nebo třívrstvé sítě. Příkladem funkce, jež náleží do $TC_3^0 \setminus TC_2^0$, je *booleovský skalární součin IP*: $\{0, 1\}^{2k} \to \{0, 1\}$ ($k \geq 1$), který definujeme jako

$$IP(x_1,\ldots,x_k,x_1',\ldots,x_k') = \bigoplus_{i=1}^{k} AND(x_i, x_i'), \tag{12.8}$$

kde $\oplus$ značí paritu k bitů, která nabývá 1, právě když je počet vstupních bitů s hodnotou 1 lichý. Tento výsledek znamená, že pro perceptronové sítě polynomiální velikosti s polynomiálními váhami mají třívrstvé sítě prokazatelně větší výpočetní sílu než dvouvrstvé. Odpovídající separace pro neomezené váhy je doposud otevřeným problémem.

Separace TC^0-hierarchie pro hloubky větší než 3 není známa a představuje jeden z největších otevřených problémů v současné teorii složitosti. V této chvíli je dokonce myslitelné (i když nepravděpodobné), že by všechny funkce ve třídě NP mohly být počítány pomocí prahových obvodů hloubky 3 s polynomiálním počtem hradel. V takovém případě by se TC^0-hierarchie zhroutila v hloubce 3, tj. $TC^0 \subseteq TC_3^0$.

12.3.1.5 Symetrické booleovské funkce

Důležitou třídou booleovských funkcí jsou *symetrické funkce*, jejichž hodnoty nezávisí na pořadí vstupních proměnných, tj. hodnota symetrické funkce závisí jen na počtu vstupních bitů s hodnotou 1. Typickým příkladem symetrické funkce je parita. Je známo, že libovolnou symetrickou funkci n proměnných je možné implementovat pomocí prahového obvodu velikosti $O(\sqrt{n})$ hradel a hloubky 3 s polynomiálními váhami (Siu a kol., 1991). Navíc velikost takového obvodu je téměř optimální, protože v nejhorším případě musí obvod obsahovat aspoň $\Omega(\sqrt{n/\log n})$ hradel, i když povolíme neomezenou hloubku a váhy. Pro hloubku 2 dokázal O'Neil (1971) přiléhavou dolní mez $\Omega(\sqrt{n})$.

Tento výsledek také dosvědčuje, že prahové obvody jsou výpočetně silnější než tzv. *obvody AND–OR*, jejichž hradla počítají jen základní logické funkce AND, OR a NOT. Bylo totiž dokázáno (Furst a kol., 1984), (Yao, 1985), (Håstad, 1989), že paritu nelze počítat pomocí obvodů AND–OR polynomiální velikosti a konstantní hloubky (odpovídajících složitostní třídě označované AC^0). To vedlo k domněnce slabšího kolapsu TC^0-hierarchie, tj. $AC^0 \subseteq TC_3^0$. Částečný výsledek v tomto směru dosáhl Allender (1989), když dokázal, že funkce z AC^0 lze počítat pomocí prahových obvodů hloubky 3 a subexponenciální velikosti $n^{\log^c n}$ pro nějakou konstantu c.

12.3.1.6 Aritmetické funkce

Výpočetní sílu prahových obvodů polynomiální velikosti a konstantní hloubky budeme dále ilustrovat na jejich schopnosti implementovat základní aritmetické funkce (Razborov, 1992), (Roychowdhury a kol., 1994), (Siu a kol., 1995). Například použitím výsledku Goldmanna a Karpinského (1998) je možné zkonstruovat dvouvrstvé perceptronové sítě polynomiální velikosti s polynomiálními váhami pro porovnání a pro součet dvou (popř. dokonce n) n-bitových binárních čísel (Alon, Bruck, 1994), (Siu, Roychowdhury, 1994). Nebo lze součin a podíl dvou n-bitových binárních čísel, n-tou mocninu (Siu a Roychowdhury, 1994) a třídění n takových čísel (Siu a kol., 1993) implementovat pomocí třívrstvých sítí.

Tabulka 12.1 Počet vrstev dopředné sítě polynomiální velikosti a vah počítající aritmetické funkce

funkce	dolní odhad	horní odhad
porovnání	2	2
součet	2	2
násobný součet	2	2
součin	3	3
dělení	3	3
mocnina	2	3
třídění	3	3
násobný součin	3	4

Z výsledku Hajnala a kol. (1993) vyplývá, že pro polynomiální velikost a váhy není možné hloubku těchto aritmetických obvodů zmenšit (Hofmeister, Pudlák, 1992), (Wegener, 1993), (Siu a kol., 1993), kromě obvodu pro mocninu, kde je tato otázka otevřená. Na druhou stranu mohou být polynomiální velikosti obvodů, které mají optimální hloubku, příliš velké na to, aby se daly ve skutečnosti realizovat. V takovém případě lze přidáním vrstvy snížit počet hradel na rozumnou míru. Například třívrstvou síť nepřijatelné polynomiální velikosti $O(n^{27}\log n)$, která násobí dvě n-bitová čísla, je možné nahradit ekvivalentním čtyřvrstvým obvodem s $O(n^2)$ prahovými hradly (Siu, Roychowdhury, 1994). V hloubce 4 lze pak také realizovat součin n n-bitových binárních čísel (Siu, Roychowdhury, 1994).

Uvedené výsledky o optimální hloubce aritmetických prahových obvodů polynomiální velikosti a vah jsou shrnuty v tabulce 12.1 (Razborov, 1992), (Hofmeister, 1994). Vyplývá z nich, že libovolnou analytickou funkci je možné s velkou přesností aproximovat pomocí perceptronové sítě polynomiální velikosti a vah s využitím jen malého konstantního počtu vrstev, pokud je reálná hodnota argumentu této funkce kódována binárně pomocí n vstupních bitů sítě (Reif, Tate, 1992).

12.3.1.7 Celková délka propojení

Teprve nedávno byla zavedena nová složitostní míra dopředných sítí, tzv. *celková délka propojení* (total wire length), viz (Legenstein, Maass, 2002), (Legenstein, Maass, 2005). Tato míra se snaží postihnout některé důležité aspekty VLSI implementace speciálních obvodů, např. neuronových sítí určených pro efektivní zpracování velkého počtu paralelních senzorových vstupů. V jednom z možných modelů se předpokládá, že hradla

jsou umístěna v průnikových bodech dvoudimenzionální mřížky, která má jednotkovou vzdálenost mezi sousedními průnikovými body. Hradla mohou být spojena v rovině libovolně pomocí vodičů, jež se mohou křížit. Celková délka propojení je pak definována jako minimální hodnota součtu délek vodičů v obvodu vzhledem ke všem možným umístěním hradel. Obecné (prahové) hradlo s k vstupy je zde modelováno jako mikroobvod (s jednotkovým časem výpočtu) pokrývající k průnikových bodů mřížky, které jsou spojeny vodičem libovolným způsobem.

Architektura dopředné sítě, která by měla optimální (např. téměř lineární) celkovou délku propojení, se musí zásadně lišit od běžně navrhovaných obvodů, protože úplné propojení mezi dvěma dvoudimenzionálními vrstvami s lineárním počtem hradel $O(n)$ vyžaduje celkovou délku propojení $\Omega(n^{2,5})$. Například booleovskou funkci $P_{LR}^{k}: \{0,1\}^{2k} \to \{0,1\}$ $(k \geq 2)$, definovanou jako $P_{LR}^{k}(x_1,..., x_k, x'_1,..., x'_k) = 1$, právě když existuje $1 \leq i < j \leq k$ takové, že $x_i = x'_j = 1$, která je prototypem detektoru jednoduchého globálního vzoru, lze počítat pomocí dvouvrstvé sítě s $2\log_2 k + 1$ prahovými hradly a celkovou délkou propojení $O(k \log k)$ (Legenstein, Maass, 2002).

12.3.2 Analogové dopředné sítě

V tomto odstavci porovnáme výpočetní vlastnosti binárních dopředných neuronových sítí a analogových acyklických sítí s logistickou sigmoidou (12.5) jako aktivační funkcí, i když následující výsledky jsou platné i pro obecnější třídy sigmoidálních hradel. Tyto analogové sítě jsou pravděpodobně nejčastěji používaným výpočetním modelem v praktických aplikacích neuronových sítí, například při učení pomocí algoritmu backpropagation (Rumelhart a kol., 1986). Booleovské prahové obvody je zřejmě možné implementovat pomocí analogových dopředných sítí stejné topologie tak, že zvýšíme absolutní hodnotu vah (vynásobením nenulové excitace (12.1) dostatečně velkým kladným číslem) v případě, že je třeba přesnější aproximace Heavisideovy funkce (12.3) pomocí logistické sigmoidy (12.5) (Maass a kol., 1991).

12.3.2.1 Konstantní velikost

Na druhou stranu použitím reálných hodnot stavů neuronů lze dosáhnout efektivnějších implementací funkcí pomocí malých analogových sítí konstantní velikosti. Například booleovskou funkci $F_k: \{0, 1\}^{2k} \to \{0, 1\}$ $(k \geq 1)$ definovanou jako

$$F_k(x_1,..., x_k, x'_1,..., x'_k) = \mathit{Majority}(x_1,..., x_k) \oplus \mathit{Majority}(x'_1,..., x'_k),$$

kde $\mathit{Majority}(x_1,..., x_k) = 1$, právě když $\sum_{i=1}^{k} x \geq \frac{k}{2}$, a $\oplus$ značí funkci XOR, lze počítat pomocí dvouvrstvé sítě skládající se z pěti analogových jednotek s omezenými váhami a s výstupní separací $\varepsilon = \Omega(k^{-2})$ (popř. $\varepsilon = \Omega(1)$, pokud povolíme polynomiální váhy). Funkci F_k však nelze implementovat pomocí dvouvrstvé binární sítě s konstantním počtem prahových hradel, i když povolíme neomezené váhy (Maass a kol., 1991). Tento výsledek dosvědčuje, že analogové dopředné sítě jsou o trochu silnějším výpočetním prostředkem než jejich binární protějšky.

Podobný výsledek, který však nezávisí na hloubce obvodu, byl dokázán pro *unární druhou mocninu* $SQ_k : \{0,1\}^{k^2+k} \to \{0,1\}$ $(k \geq 1)$ definovanou jako

$$SQ_k(x_1,\ldots,x_k,z_1,\ldots,z_{k^2}) = 1, \text{ právě když } (\sum_{i=1}^{k} x_i)^2 \geq \sum_{i=1}^{k^2} z_i \ .$$

Tuto funkci je možné počítat jen pomocí dvou analogových jednotek s polynomiálními váhami a separací $\varepsilon = \Omega(1)$, zatímco libovolná binární dopředná síť, která počítá SQ_k, má velikost aspoň $\Omega(\log k)$ prahových hradel i pro neomezenou hloubku a váhy (DasGupta, Schnitger, 1996). Tedy velikost dopředné sítě lze někdy zmenšit o logaritmický faktor, když nahradíme prahová hradla sigmoidálními.

12.3.2.2 Polynomiální velikost

Složitostní třídu TC_d^0 je možné zobecnit pro analogové sítě tak, že $\mathrm{TC}_d^0(\sigma)$ $(d \geq 1)$ obsahuje všechny funkce, které lze počítat analogovými dopřednými sítěmi polynomiální velikosti a hloubky d, s polynomiálními váhami, aktivační funkcí (12.5) a separací $\varepsilon = \Omega(1)$. Ukazuje se, že na této úrovni analogové a binární třídy splývají, tj. $\mathrm{TC}_d^0(\sigma) = \mathrm{TC}_d^0$ pro každé $d \geq 1$ (Maass a kol., 1991). Jinými slovy analogové a binární dopředné sítě polynomiální velikosti s daným počtem vrstev mají stejnou výpočetní sílu, a proto i v případě analogových sítí si třívrstvé sítě zachovávají výpočetní převahu nad dvouvrstvými. Tato výpočetní ekvivalence analogových a binárních dopředných sítí polynomiální velikosti platí i pro neomezenou hloubku a exponenciální váhy, pokud povolíme vzrůst počtu vrstev u binární sítě o konstantní faktor (DasGupta, Schnitger, 1993).

12.4 Rekurentní neuronové sítě

Architekturu *rekurentních* (*cyklických*) *neuronových sítí* tvoří obecně cyklický graf, a proto výpočty těchto sítí nemusí skončit po konečném počtu kroků. Pro rekurentní sítě lze uvažovat různé výpočetní módy (režimy) v závislosti na volbě množin aktualizovaných jednotek α_t ve vzorci (12.2). Rekurentní síť pracuje v *sekvenčním* módu, jestliže v každém časovém okamžiku $t \geq 1$ nejvýše jedna jednotka aktualizuje svůj stav podle (12.2), tj. $|\alpha_t| \leq 1$. Abychom formálně vyloučili dlouhé konstantní mezivýpočty, kdy jsou aktualizovány jen ty jednotky, které ve skutečnosti nemění své výstupy, zavádí se koncept tzv. *produktivního* výpočtu délky t^*, v jehož průběhu se v každém časovém kroku $1 \leq t \leq t^*$ aktualizuje aspoň jedna jednotka $j \in \alpha_t$ tak, že $y_j^{(t)} \neq y_j^{(t-1)}$. Produktivní výpočet rekurentní sítě *skončí*, *konverguje* nebo *dosáhne stabilního stavu* $\boldsymbol{y}^{(t^*)}$ v čase $t^* \geq 0$, jestliže $\boldsymbol{y}^{(t^*)} = \boldsymbol{y}^{(t^*+k)}$ pro všechna $k \geq 1$ (popř. pro analogové sítě platí aspoň $\left\| \boldsymbol{y}^{(t^*)} - \boldsymbol{y}^{(t^*+k)} \right\| \leq \varepsilon$ pro nějakou malou konstantu $0 \leq \varepsilon < 1$). Obvykle se předpokládá

systematická volba α_t, např. $\alpha_{\tau s+j} = \{j\}$ pro $j = 1,\ldots, s$, kde $\tau = 0, 1, 2,\ldots$ představuje diskrétní *makroskopický čas*. V rámci jednoho makroskopického kroku jsou aktualizovány všechny jednotky v síti (obecně mohou některé z nich aktualizovat svůj výstup vícekrát). Naproti tomu v *paralelním* režimu výpočtu existuje časový okamžik $t \geq 1$, v němž aspoň dva neurony současně přepočítávají svůj nový výstup, tj. $|\alpha_t| \geq 2$. Obvykle se předpokládá *plně paralelní* mód, ve kterém jsou v každém časovém kroku aktualizovány všechny jednotky, tj. $\alpha_t = V$ pro všechna $t \geq 1$.

Při *asynchronních* výpočtech je volba aktualizační množiny α_t náhodná, takže každá jednotka v síti se de facto nezávisle rozhoduje, kdy bude aktualizovat svůj stav. Oproti tomu při *synchronních* výpočtech jsou množiny α_t deterministicky určeny předem pro každý časový okamžik t. Pro binární stavy bylo ukázáno (Orponen, 1997b), že zdánlivě slabší asynchronní model má v podstatě stejnou výpočetní sílu jako jeho systematický synchronní protějšek v sekvenčním i paralelním módu dokonce i v případě symetrických sítí (viz odst. 12.4.3). Proto se dále většinou omezíme na synchronní model.

Pro účely univerzálních výpočtů nad vstupy libovolné délky byly pro rekurentní sítě zavedeny různé vstupní protokoly. V odstavci 12.4.1 popíšeme vstupní protokol pro *konečné* sítě s konečným pevným počtem jednotek, zatímco v odst. 12.4.4 se budeme zabývat nekonečnými posloupnostmi cyklických sítí, které obsahují jednu síť pro každou délku vstupu.

12.4.1 Konečné neuronové akceptory jazyků

Výpočetní síla rekurentních sítí se podobně jako u tradičních výpočetních modelů studuje tak, že se sítě používají jako akceptory jazyků $L \subseteq \{0,1\}^* = \bigcup_{n\geq 0} \{0,1\}^n$ nad binární abecedou. Jazyk L odpovídá danému rozhodovacímu problému v tom smyslu, že L se skládá ze vstupních instancí, pro které je odpověď na tento problém „ano". Například rozhodovací problém, zda je daná booleovská formule splnitelná, je reprezentován jazykem SAT obsahujícím všechny splnitelné booleovské formule. Říkáme, že jazyk $L \subseteq \{0, 1\}^*$ s charakteristickou funkcí $c_L : \{0, 1\}^* \to \{0, 1\}$ (tj. slovo $\boldsymbol{x} \in \{0, 1\}^*$ náleží do jazyka L, právě když $c_L(\boldsymbol{x}) = 1$) je *přijímán* (*rozpoznáván*) výpočetním modelem M (neboli odpovídající rozhodovací problém je řešitelný pomocí M), jestliže M parciálně implementuje c_L v tom smyslu, že výpočet M nemusí skončit pro vstupy $\boldsymbol{x} \notin L$. Dále jazyk L *je rozhodován* modelem M, jestliže M implementuje c_L totálně, např. Turingovy stroje rozhodují *rekurzivními jazyky*.

V případě konečných sítí, tzv. *neuronových akceptorů*, pracujících v plně paralelním režimu je vstupní slovo (řetězec) $\boldsymbol{x} = x_1,\ldots, x_n \in \{0, 1\}^n$ libovolné délky $n \geq 0$ prezentováno síti sekvenčně bit po bitu prostřednictvím vstupního neuronu $in \in V$. Stav této vstupní jednotky je externě nastaven na příslušné bity v předepsaných časových krocích bez ohledu na vliv zbývajících neuronů v síti. Výstupní neuron $out \in V$ následně signalizuje, zda vstupní slovo $\boldsymbol{x}$ náleží do příslušného jazyka L.

Speciálně v případě *binárních neuronových akceptorů*, založených na neuronových sítích s binárními stavy, se vstupní bity $x_1,\ldots, x_n$ předkládají síti sekvenčně s periodou $p \geq 1$ diskrétních kroků, tj. $y_{in}^{(p(i-1))} = x_i$ pro $i = 1,\ldots, n$, a výstupní neuron rozhoduje každý prefix vstupního slova on line s možným časovým zpožděním $k \geq 1$:

$$y_{\text{out}}^{(p(i-1)+k+1)} = \begin{cases} 1 & \text{jestliže } x_1 \ldots x_i \in L \\ 0 & \text{jestliže } x_1 \ldots x_i \notin L. \end{cases} \tag{12.9}$$

Pro binární sítě se dá ukázat, že konstantní časové zpoždění k je možné snížit na jednotkové za cenu jen lineárního nárůstu velikosti sítě (Šíma, Wiedermann, 1998).

V případě analogových sítí obvykle $p = 1$ a konec vstupního slova je detekován *validačním* vstupním neuronem $ival \in V$, např.

$$y_{\text{ival}}^{(t)} = \begin{cases} 1 & \text{pro } t = 0, \ldots, n-1 \\ 0 & \text{pro } t \geq n. \end{cases} \tag{12.10}$$

Výstupní neuron pak po nějakém čase $T(n) \geq n$, který závisí na délce vstupu n, hlásí výsledek rozpoznávání

$$y_{\text{out}}^{(t)} = \begin{cases} 1 & \text{jestliže } \boldsymbol{x} \in L \text{ a } t = T(n) \\ 0 & \text{v opačném případě.} \end{cases} \tag{12.11}$$

Příslušný okamžik je opět detekován validačním výstupním neuronem $oval \in V$:

$$y_{\text{oval}}^{(t)} = \begin{cases} 1 & \text{pro } t = T(n) \\ 0 & \text{pro } t \neq T(n). \end{cases} \tag{12.12}$$

V analogových sítích je možný alternativní vstupní protokol (Siegelmann, Sontag, 1995), při kterém se vstupní slovo $\boldsymbol{x}$ libovolné délky n zakóduje do reálného počátečního stavu vstupního neuronu, např. $y_{\text{in}}^{(0)} = \sum_{i=1}^{n} \frac{2x_i + 1}{4^i}$.

12.4.2 Konečné asymetrické rekurentní sítě

Výpočetní síla konečných rekurentních neuronových sítí s obecně asymetrickými váhami a saturovanou lineární aktivační funkcí (12.4) závisí na popisné složitosti vah. Příslušné výsledky jsou shrnuty v tabulce 12.2 (Siegelmann, 1994) včetně srovnání s pravděpodobnostními rekurentními sítěmi, kterými se budeme zabývat v odst. 12.5.2.

12.4.2.1 Binární sítě

Pro celočíselné váhy tyto modely splývají s konečnými binárními rekurentními sítěmi využívajícími Heavisideovu aktivační funkci (12.3). Jelikož binární neuronové akceptory složené z prahových hradel mají jen konečný počet globálních stavů, jejich výpočetní síla odpovídá konečným automatům rozpoznávajícím regulární jazyky (Kleene, 1956), viz také (Minsky, 1967), proto se jim někdy říká *neuromaty*. Při studiu efektivity neuronových implementací konečných automatů byly uvažovány jemnější míry složitosti (Alon a kol., 1991). Například daný deterministický konečný automat s q stavy je možné simulovat neuromatem velikosti $O(\sqrt{q})$ s konstantní periodou $p = 4$ předkládání vstupních bitů (Horne a Hush, 1996), (Indyk, 1995). Velikost $\Omega(\sqrt{q})$ simulujícího neuroma-

tu nelze v nejhorším případě snížit, pokud je perioda nejvýše $p = O(\log q)$ (Horne, Hush, 1996), nebo pokud předpokládáme polynomiální váhy (Indyk, 1995). Velikost neuromatů můžeme také porovnat s délkou regulárních výrazů, které mají větší popisnou sílu než deterministické konečné automaty. Libovolný regulární jazyk popsaný regulárním výrazem délky ℓ lze rozpoznat neuromatem optimální velikosti $\Theta(\ell)$ (Šíma, Wiedermann, 1998). Na druhou stranu lze sestrojit neuromat lineární velikosti rozpoznávající regulární jazyk, který lze prokazatelně popsat jen regulárním výrazem exponenciální délky, což potvrzuje velkou deskriptivní schopnost neuromatů.

Tabulka 12.2 Výpočetní síla deterministických a pravděpodobnostních analogových sítí se saturovanou lineární aktivační funkcí pracujících v diskrétním čase

Doména vah	Deterministické sítě		Doména pravděpodobností	Pravěpodobnostní sítě	
	neomezený čas	polynomiální čas		neomezený čas	polynomiální čas
celočíselné	regulární	regulární	reálné	regulární	regulární
racionální	rekurzivní	P	racionální	rekurzivní	BPP
			reálné	libovolné	Pref-BPP/log
reálné	libovolné	P/poly	reálné	libovolné	P/poly

12.4.2.2 Analogové sítě s racionálními váhami

Konečné analogové rekurentní sítě s racionálními váhami a saturovanou lineární aktivační funkcí (12.4) mohou simulovat Turingův stroj dokonce tak, že každý přechod Turingova stroje je realizován v jednom časovém kroku sítě (Siegelmann, Sontag, 1995). Aplikací tohoto výsledku na univerzální Turingův stroj dostáváme, že libovolnou funkci vyčíslitelnou na Turingově stroji v čase $T(n)$ lze počítat pomocí *fixní* analogové rekurentní sítě jen s 886 jednotkami v čase $O(T(n))$. Velikost této univerzální sítě je možné dále redukovat jen na 25 neuronů za cenu nárůstu času simulace na $O(n^2T(n))$ (Indyk, 1995). Z uvedeného výkladu vyplývá, že výpočty konečných analogových sítí s racionálními váhami v polynomiálním čase odpovídají známé složitostní třídě P. Turingovská univerzalita byla dokázána i pro obecnější třídu aktivačních funkcí (Koiran, 1996) včetně logistické sigmoidy (12.5) (Kilian, Siegelmann, 1996), i když v tomto případě známé simulace vyžadují exponenciální časovou režii k realizaci jednoho přechodu Turingova stroje.

12.4.2.3 Analogové sítě s reálnými váhami

Konečné analogové rekurentní sítě s obecnými reálnými váhami a saturovanou lineární aktivační funkcí (12.4) mohou teoreticky implementovat „superturingovské" výpočty kvůli neomezenému množství informace, kterou lze zakódovat do jejich reálných vah s neomezenou přesností. Výpočetní síla takových analogových neuronových akceptorů pracujících v čase $T(n)$ je stejná jako u nekonečných (neuniformních) posloupností prahových obvodů, jejichž velikost je polynomiální vzhledem k $T(n)$ (Siegelmann, Sontag, 1994). To znamená, že výpočty těchto sítí v polynomiálním čase odpovídají neuniformní třídě složitosti P/poly a v exponenciálním čase lze pomocí nich počítat *libovolnou* vstupně–výstupní funkci. Polynomiální výpočty konečných analogových neuronových sítí s rostoucí kolmogorovskou složitostí (objem informace) reálných vah dokonce defi-

nují vlastní hierarchii neuniformních složitostních tříd mezi P a P/poly (Balcázar a kol., 1997). Pokud se omezíme jen na rekurzivní (algoritmicky definované) váhy, analogové sítě mohou v polynomiálním čase rozhodnout právě jazyky z rekurzivní části P/poly, která je ostře větší než P. To znamená, že tento model vykazuje vyšší efektivitu i bez použití nerekurzivních vah.

12.4.2.4 Analogový šum

Všechny předchozí výsledky týkající se analogových výpočtů konečných rekurentních neuronových sítí předpokládají operace nad reálnými čísly s neomezenou přesností. Nicméně, pokud jsou výstupy jednotlivých analogových neuronů vystaveny libovolnému analogovému šumu, pak tyto sítě nejsou výpočetně silnější než neuromaty (Casey, 1996), (Maass, Orponen, 1998). Dokonce pro běžná pravděpodobnostní rozdělení šumu, která jsou nenulová na dostatečně velké části stavového prostoru, nejsou tyto sítě schopné rozpoznat všechny regulární jazyky (Maass, Sontag, 1999), (Siegelmann a kol., 2000), ale rozpoznávají spolehlivě tzv. *definitní jazyky* (Rabin, 1963). Jazyk L je definitní, jestliže existují dvě konečné množiny slov L_1 a L_2 takové, že $\boldsymbol{w} \in L$, právě když buď $\boldsymbol{w} \in L_1$, nebo $\boldsymbol{w} = \boldsymbol{v}_1\boldsymbol{v}_2$ pro nějaký prefix $\boldsymbol{v}_1$ a $\boldsymbol{v}_2 \in L_2$. Na druhou stranu, pokud je úroveň šumu omezená, pak lze spolehlivě rozpoznat všechny regulární jazyky a horní odhady na velikost příslušných neuromatů uvedené v odst. 12.4.2.1 jsou platné pro velkou třídu aktivačních funkcí σ, které mají různé konečné limity v nevlastních bodech (Maass, Orponen, 1998), (Siegelmann, 1996), (Šíma, 1997).

12.4.2.5 Problém zastavení

Pro konečné rekurentní sítě byla analyzována složitost různých důležitých rozhodovacích problémů. Například otázka, zda má daná binární síť aspoň jeden stabilní stav, je NP-úplný problém (Alon, 1985), (Godbeer a kol., 1988), (Lipscomb, 1987), (Porat, 1989). Nebo problém zastavení sítě, tj. zda rekurentní síť skončí svůj výpočet nad daným vstupem, je pro binární stavy PSPACE-úplný (Floréen, Orponen, 1994), (Lepley, Miller, 1983), (Porat, 1989). Tato otázka je algoritmicky nerozhodnutelná pro analogové sítě s racionálními váhami a 25 jednotkami, což je důsledkem turingovské výpočetní univerzality takových sítí (Indyk, 1995). Ještě poznamenejme, že výpočty rekurentních sítí velikosti s, které se vždy zastaví v čase nejvýše t^*, lze zřejmě rozvinout do obvodu velikosti st^* a hloubky t^* (Savage, 1972). To znamená, že dopředné a *konvergentní* rekurentní sítě mají principiálně stejnou výpočetní sílu (Goldschlager, Parberry, 1986).

12.4.3 Konečné symetrické rekurentní sítě

Důležitým speciálním případem rekurentních sítí jsou *symetrické*, resp. *Hopfieldovy*, *sítě*, jejichž váhy splňují $w(i, j) = w(j, i)$ pro každé $i, j \in V$, a tedy odpovídající architekturu tvoří neorientovaný graf.

12.4.3.1 Konvergence

Dynamika symetrických (Hopfieldových) sítí je podřízena tzv. *Ljapunovově*, resp. *energetické*, funkci E definované na stavovém prostoru sítě. Tato funkce je omezená a klesá podél trajektorie každého produktivního výpočtu. Z existence takové funkce vyplývá, že

síť vždy konverguje k nějakému stabilnímu stavu, který odpovídá lokálnímu minimu E. Binární symetrické sítě, které splňují $w(j, j) \geq 0$ pro každé $j \in V$ (takové sítě se někdy nazývají *semijednoduché*), vždy konvergují v sekvenčním režimu z libovolného počátečního stavu (Hopfield, 1982). Tento výsledek je možné dokázat například pomocí energetické funkce

$$E(\boldsymbol{y}) = -\sum_{j=1}^{s} y_j \left(w_{j0} + \frac{1}{2} \sum_{i=1; i \neq j}^{s} w_{ji} y_i + w_{jj} y_j \right). \tag{12.13}$$

Na druhou stranu paralelní výpočty binárních Hopfieldových sítí (včetně nesemijednoduchých) buď dosáhnou stabilního stavu, např. když je kvadratická forma (12.13) negativně definitní (Goles, 1987), (Goles-Chacc a kol., 1985), nebo nakonec střídají dva různé stavy (Bruck, Goodman, 1988), (Goles, Olivos 1981a), (Goles, 1987), (Poljak, Sůra, 1983), (Tchuente, 1986). S použitím Ljapunovovy funkce (12.13) rozšířené o aditivní člen $\sum_{j=1}^{s} \int_{0}^{y_j} \sigma^{-1}(y)\mathrm{d}y$ byly uvedené výsledky dále zobecněny pro analogové symetrické sítě s aktivační funkcí σ. Konkrétně bylo za slabého předpokladu platnosti jisté hypotézy ukázáno, že tyto sítě konvergují k pevnému bodu nebo k limitnímu cyklu délky nejvýše dvou paralelních stavových aktualizací (Fogelman-Soulié a kol., 1989), (Koiran, 1994).

12.4.3.2 Čas konvergence

Čas konvergence neboli doba výpočtu Hopfieldovy sítě je počet aktualizačních diskrétních kroků potřebných k tomu, aby síť dosáhla stabilního stavu (pro analogové sítě v rámci dané přesnosti). U symetrických sítí s s binárními neurony platí triviální horní odhad 2^s kroků, protože síť má v tomto případě jen 2^s různých stavů. V nejhorším případě může být čas konvergence opravdu exponenciální v sekvenčním (Haken, Luby, 1988) i paralelním (Goles, Olivos, 1981b) režimu. Příkladem takové symetrické sítě je sekvenční, popř. paralelní, implementace binárního čítače, který prochází většinu stavového prostoru sítě a konverguje po $\Omega(2^{s/8})$ asynchronních sekvenčních aktualizacích (Haken, 1989), popř. po $\Omega(2^{s/3})$ plně paralelních krocích (Goles, Martínez, 1989), (Goles, Martínez, 1990). Na druhou stranu byla pro binární Hopfieldovy sítě dokázána za rozumných předpokladů velmi rychlá konvergence v průměrném případě po řádově $O(\log \log s)$ paralelních krocích (Komlós, Paturi, 1988). Nicméně uvedené odhady neberou v úvahu velikost vah. Z definice energetické funkce (12.13) vyplývá horní mez $O(W)$ na čas konvergence binárních symetrických sítí pracujících v sekvenčním (Fogelman a kol., 1983), (Goles, 1985), (Goles-Chacc a kol., 1985), (Goles, Martínez, 1990), popř. paralelním (Goles, 1987), režimu, kde $W = \sum_{j,i \in V} |w_{ji}|$ je celková *váha sítě*. Například Floréen (1991) vyjádřil tento horní odhad přesněji pro celočíselné váhy. Tedy binární Hopfieldovy sítě s polynomiálními váhami konvergují v polynomiálním čase. Pro analogové Hopfieldovy sítě je možné čas konvergence vyjádřit vzhledem k počtu bitů potřebných k reprezentaci jejich vah a dá se zkonstruovat analogová symetrická síť, jejíž

výpočet skončí později než výpočet libovolné binární Hopfieldovy sítě se stejnou velikostí reprezentace (Šíma a kol., 2000).

12.4.3.3 Stabilní stavy

Při použití Hopfieldovy sítě jako asociativní paměti odpovídají stabilní stavy uloženým vzorům, proto je stanovení počtu stabilních stavů v síti důležitým problémem. Bylo ukázáno, že binární Hopfieldova síť velikosti s, jejíž váhy (kromě $w_{jj} = w_{j0} = 0, j \in V$) jsou nezávislé, identicky distribuované gaussovské náhodné veličiny s nulovou střední hodnotou, má v průměrném případě asymptoticky $1{,}05 \cdot 2^{0{,}2874s}$ stabilních stavů (McEliece a kol., 1987), (Tanaka, a Edwards, 1980). Avšak problém, zda má daná binární symetrická síť například (pokud je nesemijednoduchá) aspoň jeden (Floréen, Orponen, 1989), dva (Lipscomb, 1987) nebo tři (Floréen, Orponen, 1989) stabilní stavy, je NP-úplný. Ve skutečnosti je určení přesného počtu stabilních stavů v dané binární Hopfieldově síti algoritmicky těžkým problémem (Floréen, Orponen, 1989), (Lipscomb, 1987). Také třeba problém výpočtu *poloměru atrakce* daného stabilního stavu, což je největší počet bitů stabilního stavu, které můžeme změnit, aby příslušná Hopfieldova síť v sekvenčním nebo paralelním režimu vždy konvergovala zpět k tomuto stavu, je NP-těžký a nelze jej rozumně aproximovat v polynomiálním čase, pokud P $\neq$ NP (Floréen, Orponen, 1993).

12.4.3.4 Problém minimální energie

Hopfieldovy sítě se také používají pro heuristické řešení těžkých kombinatorických optimalizačních problémů (Hopfield, Tank, 1985). Účelová funkce optimalizačního problému je v tomto případě zakódována do energetické funkce Hopfieldovy sítě, která je minimalizována v průběhu jejího výpočtu. Proto se problém nalezení stavu Hopfieldovy sítě s minimální energií nebo energií menší, než je předepsaná hodnota, těší zvláštní pozornosti. Nicméně tento problém je obecně NP-úplný pro binární (Barahona, 1982), (Bertoni, Campadelli, 1994) i analogové (Šíma, a kol., 2000) Hopfieldovy sítě, i když pro binární sítě s planární topologií je řešitelný v polynomiálním čase (Barahona, 1982). Navíc pro binární symetrické sítě s váhou W existuje polynomiální aproximační algoritmus, který je založen na výkonném aproximačním algoritmu pro problém maximálního řezu v grafu (Goemans, Williamson, 1995), (Mahajan, Ramesh, 1999) a který řeší problém minimální energie s garantovanou absolutní chybou menší než $0{,}243W$ (Šíma a kol., 2000). Pro $W = O(s^2)$, tj. například pro Hopfieldovy sítě s s binárními neurony a konstantními váhami, tento výsledek odpovídá dolní mezi $\Omega(s^{2-\varepsilon})$pro absolutní chybu, kterou nelze garantovat žádným polynomiálním aproximačním algoritmem pro každé $\varepsilon > 0$, pokud P $\neq$ NP (Bertoni, Campadelli, 1994).

12.4.3.5 Výpočetní síla

Nakonec podáme stručný přehled výsledků o výpočetní síle Hopfieldových sítí (srovnej s odpovídajícími výsledky pro asymetrické sítě prezentovanými v odst. 12.4.2). V případě binárních stavů neuronů jsou symetrické sítě schopné simulovat libovolné *konvergentní* asymetrické sítě za cenu jen lineárního nárůstu času a velikosti simulující Hopfieldovy sítě (Orponen, 1996), (Šíma a kol., 2000). Tato efektivní obrácená implikace

slavné Hopfieldovy konvergenční věty (viz odst. 12.4.3.1) tedy ukazuje, že pro symetrické binární sítě platí ekvivalence

„konvergence ≡ symetrii vah",

tj. nejen že všechny binární symetrické sítě konvergují, ale všechny konvergentní výpočty libovolné binární asymetrické neuronové sítě mohou být efektivně implementovány pomocí Hopfieldovy sítě. Tento výsledek byl již dříve znám pro dopředné sítě, které je možné implementovat pomocí symetrických rekurentních sítí se stejnou architekturou (Parberry, 1994). Na druhou stranu symetrické neuromaty s předem neomezenou délkou vstupního slova jsou slabším výpočetním prostředkem než konečné automaty a rozpoznávají jen vlastní podtřídu regulárních jazyků, které se nazývají *Hopfieldovy jazyky* (Šíma, Wiedermann, 1998). Dá se ukázat, že regulární jazyk $L \subseteq \{0, 1\}^*$ je Hopfieldův, právě když pro každý prefix a sufix v_1, $v_2 \in \{0, 1\}^*$ a dvoubitový řetězec $x \in \{0, 1\}^2$ existuje k_0 takové, že buď $v_1x^kv_2 \in L$ pro všechna $k \geq k_0$, nebo $v_1x^kv_2 \notin L$ pro všechna $k \geq k_0$.

Také analogové symetrické neuronové akceptory jsou schopné spolehlivě rozpoznat Hopfieldovy jazyky (Šíma, 1997), což představuje dolní odhad na jejich výpočetní sílu. Vzhledem k jejich konvergenčním vlastnostem (viz odst. 12.4.3.1) je však nepravděpodobné, že by konečné analogové Hopfieldovy sítě byly schopny simulovat libovolný Turingův stroj. Nicméně, pokud k analogové symetrické síti pracující v plně paralelním režimu přidáme externí oscilátor, který generuje posloupnost bitů obsahující nekonečně mnoho tříbitových podřetězců tvaru $bx\bar{b} \in \{0,1\}^3$, kde $b \neq \bar{b}$, pak tento model může simulovat libovolnou asymetrickou rekurentní síť se stejnou kolmogorovovskou složitostí reálných vah. To znamená, že výsledky o výpočetní síle deterministických asymetrických sítí prezentované v tabulce 12.2 jsou platné i pro Hopfieldovy sítě s přidaným oscilátorem, např. pro racionální váhy jsou turingovsky univerzální. Tím dostáváme úplnou charakterizaci výpočetní síly konečných analogových neuronových sítí s racionálními váhami ve tvaru

„Turingův stroj ≡ asymetrické síti ≡ symetrické síti + oscilátoru"

spolu s nutnou a postačující podmínkou, kterou musí externí oscilátor splňovat, aby tato ekvivalence byla platná.

12.4.4 Nekonečné posloupnosti binárních rekurentních sítí

V odstavci 12.4.1 jsme se zabývali konečnými neuronovými akceptory, které vstupy s neomezenou délkou zpracovávají bit po bitu. Podobně jako u obvodů (odst. 12.3) lze použít alternativní vstupní protokol, při němž předpokládáme nekonečnou posloupnost (rodinu) $\{N_n\}$ binárních rekurentních sítí, která obsahuje jednu síť N_n pro každou délku vstupu $n \geq 0$. Tedy pro výpočty nad n-bitovými binárními vstupy $x \in \{0, 1\}^n$ se používá síť N_n, u níž jsou stavy n vstupních neuronů na začátku výpočtu (počáteční stav) nastaveny na vstup x. Pro rozpoznání jazyka $L \subseteq \{0, 1\}^*$ se pro vstupní slovo $x \in \{0, 1\}^n$ použije příslušná síť N_n, která poté, co konverguje v čase t^*, signalizuje prostřednictvím svého jediného výstupního neuronu *out*, zda x náleží do L, tj. $y_{out}^{(t^*)} = 1$, právě když $x \in L$.

Opět definujeme *velikost* $S(n)$ jako počet jednotek v N_n. Posloupnosti binárních rekurentních sítí polynomiální velikosti rozpoznávají právě jazyky ze složitostní třídy PSPACE/poly (Lepley, Miller, 1983). Ekvivalence mezi konvergentními asymetrickými a symetrickými sítěmi (odst. 12.4.3.5) implikuje stejnou výpočetní sílu pro rodiny Hopfieldových sítí polynomiální velikosti, které také rozpoznávají PSPACE/poly (Orponen, 1996). Pokud navíc Hopfieldovy sítě v těchto posloupnostech mají *polynomiální* váhy (vzhledem k délce vstupu n), pak je jejich výpočetní síla charakterizována třídou P/poly.

12.5 Pravděpodobnostní neuronové sítě

V literatuře se setkáváme s různými stochastickými variantami diskrétních perceptronových sítí, jejichž výpočetní vlastnosti byly analyzovány. Definujme referenční model *pravděpodobnostních* (*stochastických*) *sítí* tak, že k příslušné deterministické síti přidáme náhodné binární vstupní jednotky $i \in \Pi$, jejichž stavy v čase reprezentují nezávislé a identicky distribuované binární posloupnosti. Tedy pro všechny diskrétní časové okamžiky $t \geq 0$ je pravděpodobnost, že $y_i^{(t)} = 1$ pro náhodný vstupní neuron $i \in \Pi$, rovna dané hodnotě p_i ($0 \leq p_i \leq 1$), a tedy $y_i^{(t)} = 0$ nastává s pravděpodobností $1 - p_i$. Tento referenční model pravděpodobnostních sítí je obvykle srovnatelný pomocí vzájemných polynomiálních simulací s jinými typy stochastických neuronových sítí, např. se sítěmi s chybovými stavy či spoji (Siegelmann, 1999b), (von Neumann, 1956) nebo Boltzmannovými stroji (Ackley a kol., 1985), (Parberry, 1994), (Parberry, Schnitger, 1989) apod. Při studiu výpočetní síly se pravděpodobnostní sítě používají k rozpoznávání jazyků podobně jako jejich deterministické protějšky. Jazyk $L \subseteq \{0, 1\}^*$ je *ε-rozpoznáván* pravděpodobnostní neuronovou sítí, jestliže pravděpodobnost chyby sítě je nejvýše ε, kde $0 \leq \varepsilon < 1/2$, tj. pravděpodobnost, že síť odpoví 1 na vstup $\boldsymbol{x} \in \{0, 1\}^*$, je aspoň $1 - \varepsilon$ pro $\boldsymbol{x} \in L$ a nejvýše ε pro $\boldsymbol{x} \notin L$. Uvedenou symetrii v pravděpodobnosti chyby přijetí a zamítnutí vstupního slova lze vždy docílit přidáním několika náhodných vstupních jednotek (Hajnal a kol., 1993).

12.5.1 Pravděpodobnostní dopředné sítě

Pro jednoduchost nejprve uvažujme pravděpodobnostní binární dopředné sítě (pravděpodobnostní prahové obvody). Rozpoznávání jazyků pomocí posloupností pravděpodobnostních prahových obvodů s jedním výstupem, které mají velkou pravděpodobnost chyby (např. $\varepsilon = 0{,}4$), není příliš spolehlivé, avšak pravděpodobnost chyby lze libovolně snížit paralelním opakováním výpočtu. Každý jazyk, který je ε-rozpoznáván ($0 < \varepsilon < 1/2$) pravděpodobnostní dopřednou sítí hloubky d vrstev a velikosti s jednotek, může být λ-rozpoznáván stochastickým prahovým obvodem hloubky $d + 1$ a velikosti $\lceil 2\log_{4\varepsilon(1-\varepsilon)}\lambda \rceil s + 1$ hradel pro libovolné $0 < \lambda < \varepsilon$. Proto lze každou pravděpodobnostní binární dopřednou síť dokonce nahradit přijatelně velkým ekvivalentním *deterministickým* prahovým obvodem. Konkrétně Parberry a Schnitger (1989) dokázali, že pro každý jazyk $L \subseteq \{0, 1\}^n$, který je ε-rozpoznáván ($1/4 < \varepsilon < 1/2$) pravděpodobnostní dopřednou sítí s n vstupy,

hloubky d a velikosti s, existuje (deterministický) prahový obvod hloubky $d + 1$ a velikosti

$$\left\lceil \frac{8\varepsilon \ln 2}{(1-2\varepsilon)^2} + 1 \right\rceil ns + 1 \tag{12.14}$$

rozpoznávající L.

Složitostní třídu TC^0_d ($d \geq 1$) pro posloupnosti prahových obvodů (viz odst. 12.3.1.4) lze zobecnit pro stochastické sítě tak, že odpovídající pravděpodobnostní třída RTC^0_d obsahuje všechny jazyky, které jsou $\varepsilon(n)$-rozpoznávány posloupnostmi pravděpodobnostních prahových obvodů polynomiální velikosti, hloubky $d \geq 1$ a s polynomiálními váhami, u nichž je pravděpodobnost chyby pro každou délku vstupu $n \geq 0$ dána reálnou posloupností $\varepsilon(n) = \frac{1}{2} - \frac{1}{n^{O(1)}}$. Například jazyk IP, který obsahuje vstupy, pro něž má booleovský skalární součin (12.8) hodnotu 1, náleží do třídy RTC^0_2 (Hajnal a kol., 1993). To znamená, že pravděpodobnostní dopředné sítě mohou být efektivnější než jejich deterministické protějšky, protože $\mathrm{IP} \notin \mathrm{TC}^0_2$ (viz odst. 12.3.1.4). Na druhou stranu aspoň u *neuniformních* posloupností pravděpodobnostních prahových obvodů lze ušetřit nejvýše jednu vrstvu, neboť v tomto případě platí $\mathrm{RTC}^0_d \subseteq \mathrm{TC}^0_{d+1}$ pro každé $d \geq 1$ (Hajnal a kol., 1993), jak je načrtnuto na obrázku 12.2.

12.5.2 Pravděpodobnostní rekurentní sítě

Výpočetní síla konečných rekurentních sítí se saturovanou lineární aktivační funkcí (12.4) byla analyzována také pro pravděpodobnostní sítě (Siegelmann, 1999b). Výsledky jsou shrnuty a porovnány s deterministickými modely v tabulce 12.2. Pro celočíselné váhy binární pravděpodobnostní sítě ε-rozpoznávají regulární jazyky. Analogové pravděpodobnostní sítě s racionálními váhami v polynomiálním čase ε-rozpoznávají právě jazyky z neuniformní třídy složitosti Pref-BPP/log (viz příloha). Tato slabá superturingovská výpočetní schopnost je důsledkem toho, že pravděpodobnostmi p_i u náhodných vstupů $i \in \Pi$ jsou *libovolná* reálná čísla. Pokud se omezíme na racionální hodnoty pravděpodobností, pak je výpočetní síla analogových pravděpodobnostních sítí s racionálními váhami v polynomiálním čase charakterizována rekurzivní složitostní třídou BPP. Nakonec pravděpodobnostní rekurentní sítě s obecnými reálnými váhami pracující v čase $T(n)$ lze simulovat pomocí odpovídajících deterministických sítí v čase $nT(n) + n^2$. To znamená, že polynomiální výpočty těchto sítí jsou také charakterizovány složitostní třídou P/poly.

12.6 Spojitý čas

Analogový stav neuronové sítě $\boldsymbol{y}(t) \in \Re^s$ pracující ve spojitém čase (zkráceně spojité síti) je definován pro každý reálný časový okamžik $t \geq 0$ obvykle jako řešení soustavy s diferenciálních rovnic, ve které každá rovnice odpovídá jedné jednotce ve spojité síti. Počáteční podmínky této soustavy jsou dány počátečním stavem sítě $\boldsymbol{y}(0)$. Výpočetní dynamika spojité sítě může být zadána například soustavou

$$\frac{\mathrm{d}y_j}{\mathrm{d}t}(t) = -y_j(t) + \sigma(\xi_j(t)) = -y_j(t) + \sigma\left(\sum_{i=0}^{s} w_{ji} y_i(t)\right) \tag{12.15}$$

pro $j = 1,\ldots, s$, kde excitace $\xi_j(t)$ jednotky j je definována vzorcem (12.1) a σ je saturovaná lineární aktivační funkcí (12.4). Pomocí Ljapunovovy funkce

$$E(\boldsymbol{y}) = -\sum_{j=1}^{s} y_j \left(w_{j0} + \frac{1}{2}\sum_{i=1}^{s} w_{ji} y_i\right) + \sum_{j=1}^{s} \int_0^{y_j} \sigma^{-1}(y)\mathrm{d}y \tag{12.16}$$

lze ukázat, že spojitá symetrická síť ($w_{ji} = w_{ij}$) s výpočetní dynamikou (12.15) konverguje z libovolného počátečního stavu $\boldsymbol{y}$ (0) ke stabilnímu stavu, pro který platí $\frac{\mathrm{d}y_j}{\mathrm{d}t} = 0$, resp. $y_j(t) = \sigma(\xi_j(t))$, pro všechna $j = 1,\ldots, s$ (Cohen, Grossberg, 1983), (Hopfield, 1984). V nejhorším případě však může být čas konvergence exponenciální vzhledem k velikosti sítě (Šíma, Orponen, 2003b). Z hlediska výpočetní síly lze pro danou binární asymetrickou rekurentní neuronovou síť pracující v diskrétním čase zkonstruovat spojitou symetrickou síť, která ji simuluje za cenu jen lineárního nárůstu velikosti sítě (Šíma, Orponen, 2003a). To znamená, že posloupnosti spojitých (symetrických) sítí polynomiální velikosti rozpoznávají aspoň jazyky z třídy PSPACE/poly.

12.7 Závěr

V této kapitole jsme podali systematický přehled výsledků týkajících se složitostně teoretických vlastností modelů neuronových sítí, které implementují digitální výpočty. Z prostorových důvodů jsme se omezili na tradiční perceptronové jednotky, protože jejich výpočetní vlastnosti jsou nejlépe prozkoumány. Jiné typy jednotek byly z tohoto hlediska studovány teprve nedávno a jejich analýza ještě není úplná. Nicméně se ukazuje, že jednotky RBF mají srovnatelnou výpočetní sílu jako perceptrony (Šorel, Šíma, 2004), (Friedrichs, Schmitt, 2005), zatímco hradla WTA (Winner-Take-All) mohou některé funkce implementovat efektivněji (Maass, 2000). Také sítě spiking neuronů mají nepatrně vyšší efektivitu než perceptronové sítě (Maass, 1996), (Maass, 1997). Navíc spiking neurony kromě toho, že věrněji modelují biologické neurony, používají frekvenční kódování stavů, jehož význam pro efektivní počítání nepochybně zasluhuje hlubší studium.

V našem přehledu jsme se především zaměřili na modely neuronových sítí pracující v diskrétním čase a o spojitých sítích zatím víme jen to, že jsou výpočetně aspoň tak

silné jako jejich diskrétní protějšky. To je způsobeno tím, že nemáme k dispozici adekvátní teoretické nástroje (např. složitostní míry, redukce, univerzální výpočet) pro „přirozené" spojité výpočty (Maass a kol., 2002), (Moore, 1998), (Orponen 1997a). Spojité neuronové sítě mohou v tomto ohledu posloužit jako referenční modely pro vývoj takových nástrojů (Ben-Hur a kol., 2002), (Gori, Meer, 2002).

Analýza dopředných perceptronových sítí ukázala, že mnohé důležité funkce je možné implementovat pomocí jen několika vrstev a rozumného počtu neuronů. Velmi důležitým výsledkem v tomto kontextu je, že dvě vrstvy perceptronů nejsou dostatečné pro efektivní implementaci jistých funkcí, které vyžadují třívrstvé sítě. Naproti tomu výpočetní síla rekurentních sítí byla elegantně charakterizovaná pomocí deskriptivní složitosti vah, jak je shrnuto v tabulce 12.2, která navíc ukazuje, že stochastičnost hraje u modelů neuronových sítí podobnou úlohu jako u Turingových strojů. Analogové stavy rekurentních sítí mohou teoreticky kódovat nekonečné množství informace, což není z hlediska praktické implementace reálné, nicméně analogové neuronové sítě s omezenou přesností reálných parametrů mohou být oproti binárním sítím efektivnějším výpočetním prostředkem. Pro binární sítě pak umíme ukázat, že symetrie vah a konvergence sítě znamená de facto totéž.

Jak jsme ukázáli v tomto přehledu, naše poznání výpočetní síly neuronových sítí je z hlediska jejich schopnosti implementovat univerzální výpočty celkem uspokojivé. Důležitou oblastí pro další výzkum je analýza výpočetních vlastností neuronových sítí pro specializované výpočty z pohledu jejich aplikací.

Příloha

$g = \boldsymbol{O}(f(n))$: $(\exists \varepsilon > 0)\ (\exists n_0)\ (\forall n \geq n_0)\ g(n) < \varepsilon f(n)$.
$g = \boldsymbol{\Omega}(f(n))$: $(\exists \varepsilon > 0)\ (\exists n_0)\ (\forall n \geq n_0)\ g(n) > \varepsilon f(n)$.
$g = \boldsymbol{\Theta}(f(n))$: $g = O(f(n))$ a $g = \Omega(f(n))$.

P: Třída jazyků rozpoznatelných (deterministickým) Turingovým strojem v polynomiálním čase, tj. během prvních $T(n) = O(n^c)$ výpočetních kroků stroje pro nějakou pevnou konstantu c, kde n je délka vstupu (např. počet bitů jeho binární reprezentace). Třída P obsahuje problémy, které jsou považovány za výpočetně zvládnutelné.

NP: Třída jazyků rozpoznatelných nedeterministickým Turingovým strojem v polynomiálním čase. Nedeterministický Turingův stroj si může v každém výpočetním kroku zvolit svoji další akci obecně z množiny několika (bez újmy na obecnosti dvou) možností. Z toho vyplývá, že pro daný vstup neexistuje obecně jen jeden (deterministický) výpočet (cesta), ale celá množina (strom) možných výpočtů. Pro přijetí vstupního slova nedeterministickým Turingovým strojem musí existovat aspoň jeden přijímající výpočet v této množině. Jinými slovy třída NP obsahuje ty problémy, jejichž řešení lze nejprve uhádnout (odpovídá nedeterministické volbě přijímajícího výpočtu z množiny možných výpočtů) a pak ověřit jeho správnost (zda tento výpočet je opravdu přijímající) v polynomiálním čase. Například jazyk splnitelných booleovských formulí SAT je v NP, protože pro danou booleovskou formuli lze uhádnout ohodnocení příslušných proměnných a dosazením ověřit v polynomiálním čase, zda toto ohodnocení formuli splňuje.

A je **NP-těžký**: Každý problém z NP lze převést (redukovat) na A v polynomiálním čase, tj. pro každý B v NP existuje funkce f, kterou je možné počítat v polynomiálním čase a která transformuje vstup x pro problém B na nějaký vstup $f(x)$ pro problém A tak, že $x \in B$, právě když $f(x) \in A$. Tedy vyřešením jediného NP-těžkého problému v polynomiálním čase bychom uměli řešit všechny problémy v NP v polynomiálním čase (tj. P = NP).

A je **NP-úplný**: A je NP-těžký a A je v NP. Třída NP-úplných problémů tedy obsahuje nejtěžší problémy v NP, o nichž se předpokládá, že nejsou výpočetně zvládnutelné (tj. P ≠ NP). Například problém splnitelných booleovských formulí SAT je NP-úplný.

co-NP: Třída jazyků, jejichž doplňky jsou rozpoznatelné nedeterministickým Turingovým strojem v polynomiálním čase.

A je **co-NP-úplný**: A je co-NP a každý problém z co-NP lze převést na A v polynomiálním čase.

PSPACE: Třída jazyků rozpoznatelných (deterministickým) Turingovým strojem pracujícím v polynomiálním prostoru, tj. s použitím jen $S(n) = O(n^c)$ políček pásky pro nějakou pevnou konstantu c, kde n je délka vstupu. Platí P ⊆ NP ⊆ PSPACE a P ⊆ co-NP ⊆ PSPACE, ale dosud nebylo dokázáno, že tyto inkluze jsou ostré, i když se to očekává.

A je **PSPACE-úplný**: A je v PSPACE a každý problém z PSPACE lze převést na A v polynomiálním čase. Množina pravdivých kvantifikovaných booleovských formulí (bez volných proměnných) QBF je příkladem PSPACE-úplného problému.

BPP: Třída jazyků rozpoznatelných pravděpodobnostním Turingovým strojem v polynomiálním čase s pravděpodobností chyby shora omezenou nějakou kladnou konstantou $\varepsilon < 1/2$. Pravděpodobnostní Turingův stroj je nedeterministickým Turingovým strojem, jenž může v každém výpočetním kroku volit z právě dvou různých možných akcí. Navíc pro daný vstup je počet kroků ve všech možných výpočtech stejný a každý takový výpočet skončí v koncovém stavu, ve kterém vstup buď přijímá, nebo zamítá. Pro libovolný vstup x je pravděpodobnost chyby definována jako podíl počtu výpočtů nad x s chybnou odpovědí k celkovému počtu výpočtů nad x. Problémy z třídy BPP jsou považovány za algoritmicky zvládnutelné problémy a platí P ⊆ BPP ⊆ PSPACE, ale doposud neumíme dokázat, že tyto inkluze jsou ostré.

P/poly: Třída jazyků rozpoznatelných Turingovým strojem v polynomiálním čase s polynomiální poradní (advice) funkcí. Poradní funkce f, která nemusí být rekurzivní, poskytuje Turingovu stroji na začátku výpočtu externí radu ve tvaru řetězce $f(n)$, jenž závisí jen na délce vstupu n. Pro délku tohoto řetězce u polynomiálních poradních funkcí platí $|f(n)| = O(n^c)$ pro nějakou pevnou konstantu c.

PSPACE/poly: Třída jazyků rozpoznatelných Turingovým strojem s polynomiálním prostorem a polynomiální poradní funkcí.

Pref-BPP/log: Třída jazyků rozpoznatelných pravděpodobnostním Turingovým strojem v polynomiálním čase s omezenou pravděpodobností chyby a logaritmickou poradní funkcí f ($|f(n)| = O(\log n)$), která je uzavřená na prefixy, tj. $f(n_1)$ je prefixem $f(n_2)$ pro každé $n_1 < n_2$.

Literatura

Ackley D. H., Hinton G. E., Sejnowski T. J.: A learning algorithm for Boltzmann machines, *Cognitive Science*, vol. 9, no. 1, 1985, s. 147–169.

Allender E.: A note on the power of threshold circuits. In: *Proceedings of the FOCS'89 Thirty Annual Symposium on Foundations of Computer Science*. IEEE Computer Society, New York 1989, s. 580–584.

Alon N.: Asynchronous threshold networks, *Graphs and Combinatorics*, vol. 1, 1985, s. 305–310.

Alon N., Bruck J.: Explicit construction of depth-2 majority circuits for comparison and addition, *SIAM Journal on Discrete Mathematics*, vol. 7, no. 1, 1994, s. 1–8.

Alon N., Dewdney A. K., Ott T. J.: Efficient simulation of finite automata by neural nets, *Journal of the ACM*, vol. 38, no. 2, 1991, s. 495–514.

Anthony M.: Discrete Mathematics of Neural Networks: Selected Topics, *Society for Industrial and Applied Mathematics*, Filadelfie 2001.

Anthony M., Bartlett P. L.: *Neural Network Learning: Theoretical Foundations*. Cambridge University, Cambridge 1999.

Balcázar J. L., Díaz J., Gabarró J.: *Structural Complexity I.* (2. vyd.), Springer-Verlag, Berlín 1995.

Balcázar J. L., Gavaldà R., Siegelmann H. T.: Computational power of neural networks: A characterization in terms of Kolmogorov complexity, *IEEE Transactions on Information Theory*, vol. 43, no. 4, 1997, s. 1175–1183.

Barahona F.: On the computational complexity of Ising spin glass models, *Journal of Physics A: Mathematical and General*, vol. 15, no. 10, 1982, s. 3241–3253.

Ben-Hur A., Siegelmann H. T., Fishman S.: A theory of complexity for continuous time systems, *Journal of Complexity*, vol. 18, no. 1, 2002, s. 51–86.

Bertoni A., Campadelli P.: On the approximability of the energy function. In: *Proceedings of the ICANN'94 Fourth International Conference on Artificial Neural Networks*. Springer-Verlag, Berlín 1994, s. 1157–1160.

Bruck J., Goodman J. W.: A generalized convergence theorem for neural networks, *IEEE Transactions on Information Theory*, vol. 34, no. 5, 1988, s. 1089–1092.

Casey M.: The dynamics of discrete-time computation, with application to recurrent neural networks and finite state machine extraction, *Neural Computation*, vol. 8, no. 6, 1996, s. 1135–1178.

Cohen M. A., Grossberg S.: Absolute stability of global pattern formation and parallel memory storage by competitive neural networks, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, no. 5, 1983, s. 815–826.

Cover T. M.: Capacity problems for linear machines. In: Kanal L. (ed.): *Pattern Recognition*. Thompson Book Co., Washington 1968, s. 283–289.

DasGupta B., Schnitger G.: The power of approximating: A comparison of activation functions. In: Hanson S. J., Cowan J. D., Giles C. L. (eds): *Advances in Neural Information Processing Systems (NIPS'92)*. vol. 5, Morgan Kaufmann, San Mateo 1993, s. 615–622.

DasGupta B., Schnitger G.: Analog versus discrete neural networks, *Neural Computation*, vol. 8, no. 4, 1996, s. 805–818.

Demlová M., Štěpánková O.: Teorie složitosti a úlohy umělé inteligence. In: Mařík a kol.: *Umělá inteligence 3*. Academia, Praha 2001, s. 262–310.

Floréen P.: Worst-case convergence times for Hopfield memories, *IEEE Transactions on Neural Networks*, vol. 2, no. 5, 1991, s. 533–535.

Floréen P., Orponen P.: On the computational complexity of analyzing Hopfield nets, *Complex Systems*, vol. 3, no. 6, 1989, s. 577–587.

Floréen P., Orponen P.: Attraction radii in Hopfield nets are hard to compute, *Neural Computation*, vol. 5, no. 5, 1993, s. 812–821.

Floréen P., Orponen P.: *Complexity issues in discrete Hopfield networks.* Research Report, no. A–1994–4, Department of Computer Science, University of Helsinki, 1994.

Fogelman F., Goles E., Weisbuch G.: Transient length in sequential iteration of threshold functions, *Discrete Applied Mathematics*, vol. 6, no. 1, 1983, s. 95–98.

Fogelman-Soulié F., Mejia C., Goles E., Martínez S.: Energy functions in neural networks with continuous local functions, *Complex Systems*, vol. 3, no. 3, 1989, s. 269–293.

Friedrichs F., Schmitt M.: On the power of Boolean computations in generalized RBF neural networks, *Neurocomputing*, vol. 63, 2005, s. 483–498.

Furst M., Saxe J. B., Sipser M.: Parity, circuits and the polynomial-time hierarchy. *Mathematical Systems Theory*, vol. 17, no. 1, 1984, s. 13–27.

Gerstner W., Kistler W. M.: *Spiking Neuron Models: Single Neurons, Populations, Plasticity.* Cambridge University Press, Cambridge 2002.

Godbeer G. H., Lipscomb J., Luby M.: *On the computational complexity of finding stable state vectors in connectionist models (Hopfield nets).* Technical Report, no. 208/88, Department of Computer Science, University of Toronto, 1988.

Goemans M. X., Williamson D. P.: Improved approximate algorithms for maximum cut and satisfiability problems using semidefinite programming, *Journal of the ACM*, vol. 42, no. 6, 1995, s. 1115–1145.

Goldmann M., Håstad J., Razborov A.: Majority gates vs. general weighted threshold gates, *Computational Complexity*, vol. 2, no. 4, 1992, s. 277–300.

Goldmann M., Karpinski M.: Simulating threshold circuits by majority circuits, *SIAM Journal on Computing*, vol. 27, no. 1, 1998, s. 230–246.

Goldschlager L. M., Parberry I.: On the construction of parallel computers from various bases of Boolean functions, *Theoretical Computer Science*, vol. 43, no. 1, 1986, s. 43–48.

Goles E.: Dynamics of positive automata networks, *Theoretical Computer Science*, vol. 41, no. 1, 1985, s. 19–32.

Goles E.: Lyapunov functions associated to automata networks. In: Fogelman-Soulié F., Robert Y., Tchuente M. (eds): *Automata Networks in Computer Science – Theory and Applications.* Manchester University Press, Manchester 1987, s. 58–81.

Goles-Chacc E., Fogelman-Soulié F., Pellegrin D.: Decreasing energy functions as a tool for studying threshold networks, *Discrete Applied Mathematics*, vol. 12, no. 3, 1985, s. 261–277.

Goles E., Martínez S.: Exponential transient classes of symmetric neural networks for synchronous and sequential updating, *Complex Systems*, vol. 3, no. 6, 1989, s. 589–597.

Goles E., Martínez S.: *Neural and Automata Networks: Dynamical Behavior and Applications*. Kluwer Academic Publishers, Dordrecht 1990.

Goles E., Olivos, J.: Comportement periodique des fonctions a seuil binaires et applications, *Discrete Applied Mathematics*, vol. 3, no. 2, 1981a, s. 93–105.

Goles E., Olivos, J.: The convergence of symmetric threshold automata, *Information and Control*, vol. 51, no. 2, 1981b, s. 98–104.

Gori M., Meer K.: A step towards a complexity theory for analog systems, *Mathematical Logic Quarterly*, vol. 48, no. 1, 2002, s. 45–58.

Grim J.: Pravděpodobnostní neuronové sítě. In: Mařík a kol.: *Umělá inteligence 4*. Academia, Praha 2003, s. 276–312.

Hajnal A., Maass W., Pudlák P., Szegedy M., Turán G.: Threshold circuits of bounded depth, *Journal of Computer and System Sciences*, vol. 46, no. 2, 1993, s. 129–154.

Haken A.: *Connectionist networks that need exponential time to stabilize.* Unpublished manuscript, Department of Computer Science, University of Toronto, 1989.

Haken A., Luby M.: Steepest descent can take exponential time for symmetric connectionist networks, *Complex Systems*, vol. 2, no. 2, 1988, s. 191–196.

Håstad J.: Almost optimal lower bounds for small depth circuits. In: Micali S. (ed.): Advances in Computing Research, *Randomness and Computation*, vol. 5, JAI Press Inc., Greenwich CT 1989, s. 143–170.

Håstad J.: On the size of weights for threshold gates, *SIAM Journal on Discrete Mathematics*, vol. 7, no. 3, 1994, s. 484–492.

Haykin S.: *Neural Networks: A Comprehensive Foundation*. 2. vyd., Prentice-Hall, Upper Saddle River, NJ, 1999.

Hegedüs T., Megiddo N.: On the geometric separability of Boolean functions, *Discrete Applied Mathematics*, vol. 66, no. 3, 1996, s. 205–218.

Hofmeister T.: Depth-efficient threshold circuits for arithmetic functions. In: Roychowdhury V. P., Siu K.-Y., Orlitsky A. (eds): *Theoretical Advances in Neural Computation and Learning*. Kluwer Academic Publishers, Boston 1994, s. 37–84.

Hofmeister T., Pudlák P.: A proof that division is not in TC^0_2, *Research Report*, no. 447, Dortmund University, 1992.

Hopcroft J. E., Ullman J. D.: *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, Reading, MA, 1979.

Hopfield J. J.: Neural networks and physical systems with emergent collective computational abilities, *Proceedings of the National Academy of Sciences USA*, vol. 79, 1982, s. 2554–2558.

Hopfield J. J.: Neurons with graded response have collective computational properties like those of two-state neurons, *Proceedings of the National Academy of Sciences USA*, vol. 81, 1984, s. 3088–3092.

Hopfield J. J., Tank D. W.: „Neural" computation of decision in optimization problems, *Biological Cybernetics*, vol. 52, no. 3, 1985, s. 141–152.

Horne B. G., Hush D. R.: On the node complexity of neural networks, *Neural Networks*, vol. 7, no. 9, 1994, s. 1413–1426.

Horne B. G., Hush D. R.: Bounds on the complexity of recurrent neural network implementations of finite state machines, *Neural Networks*, vol. 9, no. 2, 1996, s. 243–252.

Hořejš J.: Neuronové sítě. In: Mařík a kol.: *Umělá inteligence 1*. Academia, Praha 1993, s. 217–241.

Chandra A. K., Stockmeyer L. J., Vishkin U: Constant depth reducibility, *SIAM Journal on Computing*, vol. 13, no. 2, 1984, s. 423–439.

Chytil M.: *Automaty a gramatiky.* Matematický seminář, vol. 19, SNTL, Praha 1984.

Indyk P.: Optimal simulation of automata by neural nets. In: *Proceedings of the STACS'95 Twelve Annual Symposium on Theoretical Aspects of Computer Science*, LNCS 900. Springer-Verlag, Berlín 1995, s. 337–348.

Jiřina M.: Vybrané partie z neuronových sítí. In: Mařík a kol.: *Umělá inteligence 4.* Academia, Praha 2003, s. 204–251.

Kahn J., Komlós J., Szemerédi E.: On the probability that a random $\{\pm 1\}^n$-matrix is singular, *Journal of the American Mathematical Society*, vol. 8, no. 1, 1995, s. 223–240.

Kilian J., Siegelmann H. T.: The dynamic universality of sigmoidal neural networks, *Information and Computation*, vol. 128, no. 1, 1996, s. 48–56.

Kleene S. C.: Representation of events in nerve nets and finite automata. In: Shannon C. E., McCarthy J. (eds): *Automata Studies, Annals of Mathematics Studies*. vol. 34, Princeton University Press, Princeton, NJ, 1956, s. 3–41.

Koiran P.: Dynamics of discrete time, continuous state Hopfield networks, *Neural Computation*, vol. 6, no. 3, 1994, s. 459–468.

Koiran P.: A family of universal recurrent networks, *Theoretical Computer Science*, vol. 168, no. 2, 1996, s. 473–480.

Komlós J., Paturi R.: Convergence results in an associative memory model, *Neural Networks*, vol. 1, no. 3, 1988, s. 239–250.

Kůrková V.: Aproximace funkcí neuronovými sítěmi. In Mařík a kol.: *Umělá inteligence 4.* Academia, Praha 2003, s. 254–275.

Legenstein R. A., Maass W.: Neural circuits for pattern recognition with small total wire length, *Theoretical Computer Science*, vol. 287, no. 1, 2002, s. 239–249.

Legenstein R. A., Maass W.: Wire length as a circuit complexity measure, *Journal of Computer and System Sciences*, vol. 70, no. 1, 2005, s. 53–72.

Lepley M., Miller G.: *Computational power for networks of threshold devices in asynchronous environment*. Technical Report, Department of Mathematics, MIT, Cambridge, MA, 1983.

Lipscomb J.: *On the computational complexity of finding a connectionist model's stable state vectors.* M.Sc. thesis, Department of Computer Science, University of Toronto, 1987.

Lupanov O. B.: Implementing the algebra of logic functions in terms of bounded depth formulas in the basis $+, ^*, -$, *Soviet Physics Doklady*, vol. 6, no. 2, 1961, s. 107–108.

Lupanov O. B.: Circuits using threshold elements, *Soviet Physics Doklady*, vol. 17, no. 2, 1972, s. 91–93.

Maass W.: Lower bounds for the computational power of networks of spiking neurons, *Neural Computation*, vol. 8, no. 1, 1996, s. 1–40.

Maass W.: Networks of spiking neurons: the third generation of neural network models, *Neural Networks*, vol. 10, no. 9, 1997, s. 1659–1671.

Maass W.: On the computational power of Winner–Take–All, *Neural Computation*, vol. 12, no. 11, 2000, s. 2519–2536.

Maass W., Natschläger T., Markram H.: Real-time computing without stable states: A new framework for neural computation based on perturbations, *Neural Computation*, vol. 14, no. 11, 2002, s. 2531–2560.

Maass W., Orponen P.: On the effect of analog noise in discrete-time analog computations, *Neural Computation*, vol. 10, no. 5, 1998, s. 1071–1095.

Maass W., Schnitger G., Sontag E. D.: On the computational power of sigmoid versus Boolean threshold circuits. In: *Proceedings of the FOCS'91 Thirty-Second Annual Symposium on Foundations of Computer Science*. IEEE Press, New York 1991, s. 767–776.

Maass W., Sontag E. D.: Analog neural nets with Gaussian or other common noise distribution cannot recognize arbitrary regular languages, *Neural Computation*, vol. 11, no. 3, 1999, s. 771–782.

Mahajan S., Ramesh, H.: Derandomizing approximation algorithms based on semidefinite programming, *SIAM Journal on Computing*, vol. 28, no. 5, 1999, s. 1641–1663.

McEliece R. J., Posner E. C., Rodemich E. R., Venkatesh S. S.: The capacity of the Hopfield associative memory, *IEEE Transactions on Information Theory*, vol. 33, no. 4, 1987, s. 461–482.

Minsky M. L.: *Theory of neural-analog reinforcement systems and its application to the brain-model problem.* Ph.D. Thesis, Princeton University, Princeton, NJ, 1954.

Minsky M. L.: *Computation: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffs, NJ, 1967.

Minsky M. L., Papert S. A.: *Perceptrons.* The MIT Press, Cambridge, MA, 1969.

Moore C.: Finite-dimensional analog computers: flows, maps, and recurrent neural networks. In: *Proceedings of the First International Conference on Unconventional Models of Computation*. Springer-Verlag, Berlín 1998, s. 59–71.

Muroga S.: *Threshold Logic and its Applications.* Wiley-Interscience, New York 1971.

Muroga S., Toda I., Takasu S.: Theory of majority decision elements, *Journal of the Franklin Institute*, vol. 271, 1961, s. 376–418.

Nechiporuk E. I.: The synthesis of networks from threshold elements, *Problemy Kibernetiki*, vol. 11, 1964, s. 49–62.

O'Neil P. E.: Hyperplane cuts of an n-cube, *Discrete Mathematics*, vol. 1, no. 2, 1971, s. 193–195.

Orponen P.: Computational complexity of neural networks: A survey, *Nordic Journal of Computing*, vol. 1, no. 1, 1994, s. 94–110.

Orponen P.: The computational power of discrete Hopfield nets with hidden units, *Neural Computation*, vol. 8, no. 2, 1996, s. 403–415.

Orponen P.: A survey of continuous-time computation theory. In: Du D.-Z., Ko K.-I. (eds): *Advances in Algorithms, Languages, and Complexity*. Kluwer Academic Publishers, Dordrecht 1997a, s. 209–224.

Orponen P.: Computing with truly asynchronous threshold logic networks, *Theoretical Computer Science*, vol. 174, no. 1–2, 1997b, s. 123–136.

Orponen P.: The computational power of continuous time neural networks. In: *Proceedings of the SOFSEM'97 Twenty-Seventh Seminar on Current Trends in Theory and Practice of Informatics*. LNCS 1338, Springer-Verlag, Berlín 1997c, s. 86–103.

Papadimitriou C. H.: *Computational Complexity*. Addison-Wesley, Reading, MA, 1994.

Parberry I.: *Circuit Complexity and Neural Networks.* The MIT Press, Cambridge, MA, 1994.

Parberry I., Schnitger G.: Relating Boltzmann machines to conventional models of computation, *Neural Networks*, vol. 2, no. 1, 1989, s. 56–67.

Poljak S., Sůra M.: On periodical behaviour in societies with symmetric influences, *Combinatorica*, vol. 3, no. 1, 1983, s. 119–121.

Porat S.: Stability and looping in connectionist models with asymmetric weights, *Biological Cybernetics*, vol. 60, 1989, s. 335–344.

Rabin M.: Probabilistic automata, *Information and Control*, vol. 6, no. 3, 1963, s. 230–245.

Razborov A. A.: On small depth threshold circuits. In: *Proceedings of the SWAT'92 Third Scandinavian Workshop on Algorithm Theory*. LNCS 621, Springer-Verlag, Berlín 1992, s. 42–52.

Reif J. H., Tate S. R.: On threshold circuits and polynomial computations, *SIAM Journal on Computing*, vol. 21, no. 5, 1992, s. 896–908.

Rosenblatt F.: The perceptron: A probabilistic model for information storage and organization in the brain, *Psychological Review*, vol. 65, no. 6, 1958, s. 386–408.

Roychowdhury V. P., Siu K.-Y., Orlitsky A. (eds): *Theoretical Advances in Neural Computation and Learning*. Kluwer Academic Publishers, Boston 1994.

Rumelhart D. E., Hinton G. E., Williams R. J.: Learning representations by back-propagating errors, *Nature*, vol. 323, 1986, s. 533–536.

Savage J. E.: Computational work and time on finite machines, *Journal of the ACM*, vol. 19, no. 4, 1972, s. 660–674.

Schläfli L: *Theorie der vielfachen Kontinuität.* Zürcher & Furrer, Curych 1901.

Siegelmann H. T.: On the computational power of probabilistic and faulty neural networks. In: *Proceedings of the ICALP'94 Twenty-First International Colloquium on Automata, Languages, and Programming*. LNCS 820, Springer-Verlag, Berlín 1994, s. 23–34.

Siegelmann H. T.: Recurrent neural networks and finite automata, *Journal of Computational Intelligence*, vol. 12, no. 4, 1996, s. 567–574.

Siegelmann H. T.: *Neural Networks and Analog Computation: Beyond the Turing Limit*. Birkhäuser, Boston 1999a.

Siegelmann H. T.: Stochastic analog networks and computational complexity, *Journal of Complexity*, vol. 15, no. 4, 1999b, s. 451–475.

Siegelmann H. T., Roitershtein A., Ben-Hur A.: Noisy neural networks and generalizations. In: Solla S. A., Leen T. K., Müller K.-R. (eds): *Advances in Neural Information Processing Systems (NIPS'99)*. vol. 12, The MIT Press, Cambridge, MA, 2000, s. 335–341.

Siegelmann H. T., Sontag E. D.: Analog computation via neural networks, *Theoretical Computer Science*, vol. 131, no. 2, 1994, s. 331–360.

Siegelmann H. T., Sontag E. D.: Computational power of neural networks, *Journal of Computer System Science*, vol. 50, no. 1, 1995, s. 132–150.

Siu K.-Y., Bruck J., Kailath T., Hofmeister T.: Depth efficient neural networks for division and related problems, *IEEE Transactions on Information Theory*, vol. 39, no. 3, 1993, s. 946–956.

Siu K.-Y., Roychowdhury V. P.: On optimal depth threshold circuits for multiplication and related problems, *SIAM Journal on Discrete Mathematics*, vol. 7, no. 2, 1994, s. 284–292.

Siu K.-Y., Roychowdhury V. P., Kailath T.: Depth-size tradeoffs for neural computation, *IEEE Transactions on Computers*, vol. 40, no. 12, 1991, s. 1402–1412.

Siu K.-Y., Roychowdhury V. P., Kailath T.: *Discrete Neural Computation: A Theoretical Foundation*. Prentice Hall, Englewood Cliffs, NJ, 1995.

Šíma J.: *Analog* stable simulation of discrete neural networks, *Neural Network World*, vol. 7, no. 6, 1997, s. 679–686.

Šíma J.: Training a single sigmoidal neuron is hard, *Neural Computation*, vol. 14, no. 11, 2002, s. 2709–2728.

Šíma J.: Energy-based computation with symmetric Hopfield nets. In: Ablameyko S., Gori M., Goras L., Piuri V. (eds): Limitations and Future Trends in Neural Computation. *NATO Science Series: Computer & Systems Sciences*, vol. 186, IOS Press, Amsterdam 2003, s. 45–69.

Šíma J., Neruda R.: *Teoretické otázky neuronových sítí*. Matfyzpress, Praha 1996.

Šíma J., Orponen P.: Continuous-time symmetric Hopfield nets are computationally universal, *Neural Computation*, vol. 15, no. 3, 2003a, s. 693–733.

Šíma J., Orponen P.: Exponential transients in continuous-time Liapunov Systems, *Theoretical Computer Science*, vol. 306, no. 1–3, 2003b, s. 353–372.

Šíma J., Orponen P.: General-purpose computation with neural networks: A survey of complexity theoretic results, *Neural Computation*, vol. 15, no. 12, 2003c, s. 2727–2778.

Šíma J., Orponen P., Antti-Poika T.: On the computational complexity of binary and analog symmetric Hopfield nets, *Neural Computation*, vol. 12, no. 12, 2000, s. 2965–2989.

Šíma J., Wiedermann J.: Theory of neuromata, *Journal of the ACM*, vol. 45, no. 1, 1998, s. 155–178.

Šorel M., Šíma J.: Robust RBF finite automata, *Neurocomputing*, vol. 62, 2004, s. 93–110.

Tanaka F., Edwards S. F.: Analytic theory of the ground state properties of a spin glass: I. Ising spin glass, *Journal of Physics F: Metal Physics*, vol. 10, 1980, s. 2769–2778.

Tchuente M.: Sequential simulation of parallel iterations and applications, *Theoretical Computer Science*, vol. 48, no. 2–3, 1986, s. 135–144.

Uchizawa K., Douglas R., Maass W.: Energy complexity and entropy of threshold circuits. In: *Proceedings of the ICALP 2006 Thirty-Third International Colloquium on Automata, Languages and Programming*. Part I, LNCS 4051, Springer-Verlag, Berlín 2006, s. 631–642.

Vidyasagar M.: *A Theory of Learning and Generalization (With Applications to Neural Networks and Control Systems)*. Springer-Verlag, Londýn 1997.

Vollmer H.: *Introduction to Circuit Complexity*. Springer-Verlag, Berlín 1999.

von Neumann J.: The general and logical theory of automata. In: Jeffress L. A. (ed.): *Cerebral Mechanisms in Behavior*. Wiley, New York 1951, s. 1–41.

von Neumann J.: Probabilistic logics and the synthesis of reliable organisms from unreliable components. In: Shannon C. E., McCarthy J. (eds): Automata Studies, *Annals of Mathematics Studies*, vol. 34, Princeton University Press, Princeton, NJ, 1956, s. 43–98.

Wegener I.: *The Complexity of Boolean Functions*. Wiley/Teubner, Chichester 1987.

Wegener I.: Optimal lower bounds on the depth of polynomial-size threshold circuits for some arithmetic functions, *Information Processing Letters*, vol. 46, no. 2, 1993, s. 85–87.

Wiedermann J.: Complexity issues in discrete neurocomputing, *Neural Network World*, vol. 4, no. 1, 1994, s. 99–119.

Yao A. C.-C.: Separating the polynomial – time hierarchy by oracles. In: *Proceedings of the FOCS'85 Twenty-Sixth Annual Symposium on the Foundations of Computer Science*. IEEE Computer Society, New York 1985, s. 420–428.

13 Moderní metody výběru příznaků ve statistickém rozpoznávání

13.1 Úvod

Obor *rozpoznávání obrazů* či prostě *rozpoznávání* může být s určitým zjednodušením charakterizován jako řešení problému klasifikace či reprezentace dat popisujících zkoumané objekty reálného světa pomocí vektorů příznaků. Redukce dimenzionality (dimensionality reduction DR) je prováděna buď v podobě extrakce, nebo selekce (výběru) příznaků, která optimalizuje nějaké vhodné kritérium. Výběr příznaků je jedním z klíčových postupů předzpracování dat používaný při řešení nejrůznějších úloh ve statistickém rozpoznávání, strojovém učení, zpracování obrazové informace, klasifikaci dokumentů, dobývání znalostí z rozsáhlých databází atd. Smyslem redukce dimenzionality je nejen ušetřit čas a prostor vyřazením nepodstatných částí dat, ale i zlepšit úspěšnost klasifikace či přesnost reprezentace dat v budovaném systému potlačením vlivu šumových či jinak neinformativních příznaků.

13.2 Redukce dimenzionality

Termínem „obraz" budeme označovat D-rozměrný reálný vektor $\boldsymbol{x} = (x_1,..., x_D) \in \mathsf{X} \subseteq \mathsf{R}^D$, jehož prvky jsou měření odpovídající vlastnostem reprezentovaného objektu. Vektor $\boldsymbol{x}$ budeme rovněž nazývat vektorem příznaků. Příznaky jsou veličiny specifikované pro daný problém odborníkem. Na počátku specifikované příznaky by měly pokrýt maximum zjistitelné informace o uvažovaných objektech – možnou nadbytečnou informaci lze dodatečně redukovat pomocí metod popisovaných v této kapitole. Informaci od počátku chybějící však nahradit nelze. Ve většině praktických úloh proto očekáváme vysokou vstupní dimenzionalitu.

V kontextu statistického rozpoznávání nejčastěji předpokládáme, že objekt reprezentovaný obrazem $\boldsymbol{x}$ má být klasifikován do jedné z konečného počtu C různých tříd $\Omega = \{\omega_1,..., \omega_C\}$. Obraz $\boldsymbol{x} \in \mathsf{X}$ patřící do třídy ω_i považujeme za realizaci náhodného vektoru vybranou náhodně v souladu s apriorními pravděpodobnostmi $P(\omega_i)$ a podmíněnými hustotami pravděpodobnosti $p(\boldsymbol{x}|\omega_i)$, $\omega_i \in \Omega$.

Smyslem redukce dimenzionality je tedy nalézat automaticky d nových příznaků na základě vstupních D měření (pokud možno $d \ll D$) tak, aby bylo maximalizováno (či minimalizováno) vhodné kritérium informativnosti příznaků.

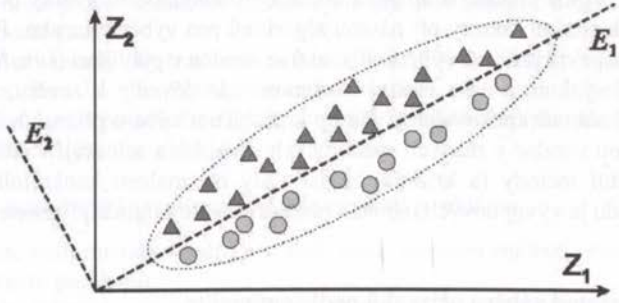
13.2.1 Redukce dimenzionality podle charakteru výsledných příznaků

Z praktických důvodů rozlišujeme dva základní přístupy k DR:
– redukci dimenzionality pomocí extrakce příznaků (feature extraction, FE),
– redukci dimenzionality pomocí výběru příznaků (feature selection, FS).
Extrakcí příznaků rozumíme proces, při němž je výsledný vektor získán jakoukoliv transformací z vektoru původního. Výsledné příznaky proto mohou reprezentovat kombinaci informace všech původních příznaků a mohou být zcela odlišně interpretovatelné. *Výběrem příznaků* rozumíme speciální případ extrakce, kdy je jednoduše vybrána podmnožina původních příznaků. Ačkoliv extrakce v obecnosti umožňuje přesnější reprezentaci dat v podprostoru, výběr bývá často upřednostněn z praktických důvodů. Je jednodušší, zachovává původní význam a interpretovatelnost měření a umožňuje úspory nákladů při akvizici dat (např. v medicíně je vhodné rozeznat, která měření netřeba u pacientů nadále provádět, aniž by se zhoršila úspěšnost automatického diagnostického systému). V dalším výkladu se zaměříme výhradně na problém *výběru příznaků*.

13.2.2 Redukce dimenzionality podle cíle

Alternativně lze rozlišit dva přístupy k redukci dimenzionality podle cíle:
– redukci dimenzionality pro optimální reprezentaci dat,
– redukci dimenzionality pro klasifikaci.

V prvním případě je cílem co nejlépe zachovat v méně rozměrném prostoru informaci obsaženou v původních datech. V druhém případě je cílem maximalizovat rozlišitelnost různých tříd.



Obr. 13.1 Redukce dimenzionality pro optimální reprezentaci dat nemusí být optimální pro rozlišení tříd.

Na obrázku 13.1 ilustrujeme, že výsledek DR pro optimální reprezentaci dat nemusí být vhodný pro rozlišení tříd. Kolečka a trojúhelníky reprezentují vzorky dvou různých tříd v dvojrozměrném prostoru. Aplikujeme-li na tento příklad metodu hlavních komponent – principal component analysis, PCA, viz např. (Duda a kol., 2000), budou data v jednorozměrném prostoru optimálně reprezentována osou E_1. Tato metoda DR vhodná pro reprezentaci však není vhodná pro klasifikaci, neboť pro odlišení tříd je v tomto případě optimální osa E_2.

V dalším textu se soustředíme především na problém klasifikace. Pro tento účel lze DR provádět *lokálně*, tj. pro každou jednotlivou třídu zvlášť, nebo *globálně*, tj. společně

pro všechny třídy. Pro obsáhlejší přehled problematiky odkazujeme na práce (Duda a kol., 2000), (Webb, 2002), (McLachlan, 2004), (Ripley, 2005) a (Theodoridis a kol., 2006).

13.3 Výběr podmnožiny příznaků

Je dána množina X_D o D příznacích. Označme X_d množinu všech možných podmnožin množiny X_D kardinality d, kde d reprezentuje požadovaný počet příznaků. Nechť $J(X)$ je kriteriální funkce, která vyhodnocuje podmnožinu příznaků $X \in \mathsf{X}_d$. Bez ztráty obecnosti předpokládejme, že větší hodnota kritéria J indikuje lepší podmnožinu příznaků. Potom lze problém FS zformulovat tak, že najdeme podmnožinu $\tilde{X}_d$, pro kterou platí

$$J(\tilde{X}_d) = \max_{X \in \mathsf{X}_d} J(X) . \tag{13.1}$$

Za předpokladu, že je k dispozici vhodná funkce J k ohodnocování efektivity podmnožin příznaků, je problém výběru příznaků redukován na vyhledávací problém podmnožiny maximalizující dané kritérium. Poznamenejme, že volba počtu příznaků d může být složitým problémem závisejícím na charakteru úlohy, pokud není optimalizace hodnoty d součástí vyhledávacího procesu.

Proces výběru příznaků probíhá ve čtyřech základních krocích: *vytváření podmnožin*, *ohodnocování podmnožin*, *ukončení procesu na základě zvoleného stop-kritéria* a *ověřování výsledků*. Volba vhodné strategie a kritéria pro ohodnocení kvality podmnožin příznaků jsou základními faktory při návrhu algoritmů pro výběr příznaků. Podrobný přehled různých aspektů procesu výběru příznaků je uveden v publikaci (Liu, Yu, 2005).

Dosavadní výzkum a naše vlastní zkušenosti nás dovedly k závěru, že neexistuje jediný vhodný obecně aplikovatelný přístup k problému výběru příznaků. Různé existující metody jsou vhodné v různých specifických situacích a selhávají v situacích jiných. Volba konkrétní metody (a kritéria) závisí vždy na znalosti konkrétního problému. Z tohoto důvodu je vývoj nových metod a příslušných paradigmat předmětem neustálého zájmu.

13.3.1 Volba metod výběru příznaků podle optimality

Metody výběru příznaků lze rozdělit na dvě skupiny podle záruky optimality výsledku:

Optimální metody. Tyto metody zahrnují například *úplné prohledání*, které je vhodné pouze pro úlohy o nízké dimenzionalitě, a *akcelerační metody* často založené na principu větví a mezí (Branch & Bound), jež jsou schopny účinně zredukovat počet testovaných kombinací d příznaků, kladou však omezující podmínky na použité kritérium. Všechny optimální metody lze považovat za značně pomalé pro řešení problémů o vysoké dimenzionalitě, viz oddíl 13.4.

Suboptimální metody. Tyto metody jsou kompromisem mezi rychlostí vyhledávání a optimalitou řešení. Zahrnují mimo jiné individuální vyhodnocování příznaků, nejrůznější gradientní metody, randomizované i deterministické, popř. kombinované, metody, viz např. (Devijver a kol., 1982), (Pudil a kol., 1994a), (Hussein a kol., 2001)

a (Somol a kol., 2008b). Ačkoliv u suboptimálních metod není zaručena optimalita výsledku, jejich výhody obvykle nad tímto nedostatkem značně převažují. Optimalita výsledku vzhledem ke zvolenému kritériu navíc nemusí mít přímý vztah k úspěšnosti klasifikace finálního klasifikátoru (Raudys, 2006). Pro vysokodimenzionální problémy jsou suboptimální metody s ohledem na výpočetní složitost jedinou volbou, viz oddíl 13.5.

13.3.2 Volba metod výběru příznaků podle způsobu vyhodnocování kritéria

S ohledem na způsob ohodnocování testovaných podmnožin příznaků můžeme metody výběru příznaků rozdělit do dalších skupin:

Filtry (*Filters*), viz (Devijver kol., 1982), (Kohavi a kol., 1997), (Dash a kol., 2002) a (Yu a kol., 2003), používají k ohodnocení podmnožiny příznaků jako kritéria míry vzdálenosti, informační míry, míry závislosti a míry konzistence, jejichž hodnoty jsou vypočteny přímo použitím trénovacích dat, viz oddíl 13.5.

Pouzdra (*Wrappers*), viz (Kohavi a kol., 1997), vyžadují předem učící algoritmus – jeho účinnost je přímo kritériem pro ohodnocení kvality vybrané podmnožiny. Obecně mají tyto metody oproti filtrům větší účinnost pro daný učící algoritmus, ale jsou výpočetně složitější a jimi vybrané příznaky mohou být špatné v obecnějším kontextu, viz oddíl 13.5.

Integrované metody (*Embedded methods*), viz (Guyon a kol., 2003), ale také (Kononenko, 1994) nebo (Pudil a kol., 1995) a (Novovičová a kol. 1996), integrují proces výběru příznaků do procesu odhadu modelu. Návrh modelu a výběr příznaků jsou tudíž neoddělitelné učící procesy, na které je možné pohlížet jako na speciální tvar metody pouzder. Integrované metody tedy nabízejí účinnost, jež konkuruje metodě pouzder, umožňují rychlejší proces učení, ale dávají výsledky úzce svázané s konkrétním modelem, viz také oddíl 13.8.

Hybridní metody (*Hybrid methods*), viz (Das, 2001), (Sebban a kol., 2002) (Liu a kol., 2005) a (Somol a kol., 2006), kombinují uvedené metody za účelem dosažení co nejlepší účinnosti určitého učícího algoritmu (např. klasifikátoru) při nízké časové složitosti srovnatelné s filtry, viz oddíl 13.7.

13.3.3 Volba metod výběru příznaků podle znalosti problému

Alternativně lze volit metody výběru příznaků podle *apriorní* znalosti pravděpodobnostních charakteristik problému:

Základní informace je známa – neboli je možné alespoň předpokládat, že podmíněné hustoty pravděpodobnosti jsou unimodální. V takovém případě můžeme postupovat odhadem parametrů vhodného modelu a následnou maximalizací některé pravděpodobnostní míry vzdálenosti (Mahalanobisovy, Bhattacharyovy atd.) za pomoci některé optimální či suboptimální metody, obvykle typu filtr (Devijver a kol., 1982).

Žádná informace není známa – neboli jediným zdrojem informace jsou trénovací data samotná. Nemůžeme předpokládat unimodalitu podmíněných hustot pravděpodobnosti, nebo naopak předpokládáme jejich komplikovanou strukturu. V takovém případě může být výhodné zvolit vhodnou metodu typu pouzdro, ale v některých případech (zejména při dostatečné velikosti dat) je možné využít alternativní metody výběru příznaků založené na modelování struktury dat za použití směsi hustot pravděpodobnosti speciálního typu. Směsový model umožňuje zachytit velmi přesně vlastnosti dat a impli-

kuje též formu klasifikačního pravidla. O směsových metodách diskutujeme v oddílu 13.8.

13.4 Optimální vyhledávací metody

Problém *optimálního* výběru příznaků (jako i obecnosti hledání optimální podmnožiny) je velmi obtížný, především vzhledem k výpočetní náročnosti. Vyhledání optimální podmnožiny příznaků mezi všemi podmnožinami dané kardinality je kombinatorický problém nezvládnutelný úplným prohledáním všech konfigurací již při relativně malém počtu uvažovaných příznaků. Již delší dobu je proto tomuto problému věnována značná pozornost ve snaze vyhledávací proces urychlit. Urychlení bylo doposud dosaženo různými způsoby, a to buď za cenu uvolnění podmínky optimality řešení (Devijver a kol., 1982), (Kirkpatrick a kol., 1983), (Caruana a kol., 1994), (Jain a kol., 1997), (Chaikla a kol., 1999) a (Kudo a kol., 2000), nebo definicí různých pomocných heuristik umožňujících identifikovat takové části vyhledávacího prostoru, které mohou být z vyhledávání vyloučeny bez ohrožení optimality výsledku (Devijver a kol., 1982) a (Fukunaga, 1990). Mezi metodami schopnými redukovat prostor hledání bez ztráty optimality lze za nejdůležitější označit algoritmus Branch & Bound (algoritmus větví a mezí) a algoritmy od něj odvozené. Idea algoritmu větví a mezí je dobře známa i mimo kontext výběru příznaků a je považována za jeden ze základních nástrojů v oblasti umělé inteligence (Lawler a kol., 1966), (Nilsson, 1971) (Kumar a kol., 1983), (Mitschele-Thiel, 1994), (Webb, 1995), (Nilsson, 1998) a (Korf, 1999). Poprvé byl algoritmus použit pro výběr příznaků v práci (Narendra a kol., 1977). V dalších letech byl v tomto kontextu detailněji zkoumán a doplňován (Fukunaga, 1990), (Hamamoto a kol., 1990), (Yu a kol., 1993), (Kudo a kol., 2000), (Chen, 2003) (Somol a kol., 2004) a (Nakariyakul a kol., 2007).

Rodina algoritmů větví a mezí řeší problém optimálního výběru d příznaků z původní množiny D příznaků za předpokladu, že kriteriální funkce J splňuje podmínku monotónnosti. Buď $\overline{\chi}_j$ množina příznaků vzniklá odebráním j příznaků $y_1, y_2, \ldots, y_j$ z množiny X_D všech D příznaků neboli

$$\overline{\chi}_j = X_D \setminus \{y_1, y_2, \ldots, y_j\}, \quad j = 1, 2, \ldots, D-1 .$$

Podmínka monotónnosti vyžaduje, aby pro podmnožiny příznaků $\overline{\chi}_1, \overline{\chi}_2, \ldots, \overline{\chi}_j$, kde

$$\overline{\chi}_1 \supset \overline{\chi}_2 \supset \ldots \supset \overline{\chi}_j ,$$

kriteriální funkce J splňovala

$$J(\overline{\chi}_1) \geq J(\overline{\chi}_2) \geq \ldots \geq J(\overline{\chi}_j) . \tag{13.2}$$

Využitím podmínky monotónnosti je možné zabránit průchodu některými částmi vyhledávacího prostoru, a tím zredukovat počet vyhodnocování kriteriální funkce, a tedy zkrátit potřebný výpočetní čas. Ačkoliv má i v nejhorším případě algoritmus větví a mezí exponenciální časovou náročnost, v praxi rychlostí zásadně překonává úplné vyhledávání. Značné úsilí bylo v posledních letech věnováno pokusům o dodatečná

zrychlení tohoto hledacího schématu. Byla navržena řada modifikovaných verzí algoritmu přinášejících zrychlení nejčastěji pro různé specifické problémy (Foroutan a kol., 1987), (Mitschele-Thiel, 1994), (Koller a kol., 1996) a (Liu a kol., 1998). Algoritmus byl paralelizován (Gengler a kol., 1994), (Xu a kol., 1995), (Yang a kol., 1999) a (Iamnitchi a kol., 2000). Zároveň byl zkoumán sám pojem optimality výběru příznaků, což vedlo k rozlišení hledacích algoritmů na *filtry* a *pouzdra* (Kohavi a kol., 1997), viz též odst. 13.3.2. Nejvýraznějšího urychlení algoritmu větví a mezí bylo v poslední době dosaženo zavedením pomocného predikčního mechanismu, který dovoluje nahradit řadu pomocných výpočtů hodnot kritéria jednoduchým odhadováním hodnot přibližných, aniž by ovšem byla ohrožena optimalita celkového výsledku (Somol a kol., 2004). Dvěma moderním algoritmům z této skupiny se budeme podrobněji věnovat v odstavcích 13.4.3 a 13.4.4.

Tabulka 13.1 Hlavní charakteristiky diskutovaných algoritmů optimálního výběru příznaků

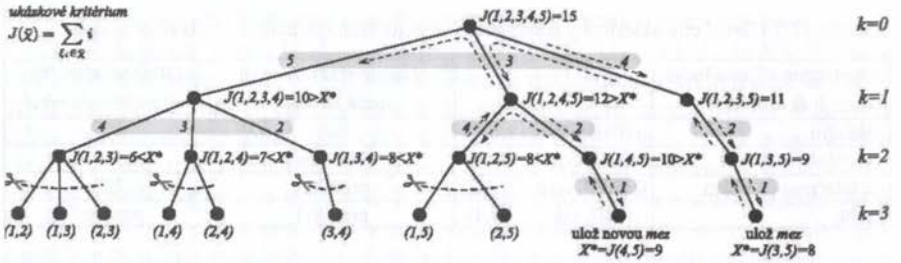
Algoritmus větví a mezí (Branch & Bound)		*Horizontální řazení uzlů stromu*	*Ušetření výpočtů při průchodu větví*
základní	(BBB, odst. 13.4.1)	NE	NE
vylepšený	(IBB, odst. 13.4.1)	výpočtem	NE
s částečnou predikcí	(BBPP, odst. 13.4.3)	predikcí	NE
rychlý	(FBB, odst. 13.4.4)	predikcí	predikcí

13.4.1 Základní pojmy a varianty algoritmu větví a mezí

Přibližme nejprve základní princip všech vyhledávacích algoritmů větví a mezí. Algoritmus konstruuje vyhledávací strom, v němž kořen reprezentuje množinu všech příznaků D a listy vyjadřují podmnožiny požadované cílové kardinality d. Procházeje stromem od kořene k listu, algoritmus postupně odebírá po jednom příznaku z momentální množiny „kandidátů" ($\overline{\chi}_k$ v k-té úrovni stromu). Algoritmus udržuje informaci o doposud nejlepší (v listu) nalezené podmnožině $\mathcal{X}$ a odpovídající hodnotě kritéria X^* (tato hodnota je onou *mezí* [bound] z názvu algoritmu). Kdykoliv v některém vnitřním uzlu stromu klesne aktuální hodnota kritéria pod tuto mez, podmínka monotónnosti (13.2) dovolí odříznout celý podstrom daného uzlu, v němž již nemůže dojít ke zvýšení meze neboli k nalezení lepšího řešení. Tímto způsobem je ušetřena značná část výpočtů spojených s vyhodnocováním kriteriální funkce. Běh algoritmu větví a mezí je demonstrován na obrázku 13.2 pro ukázkový problém výběru $d = 2$ příznaky z celkového počtu $D = 5$. Čárkované šipky naznačují postup průchodu stromem. Pro podrobnější výklad odkazujeme na práce (Devijver a kol., 1982), (Fukunaga, 1990) a (Narendra a kol., 1977).

Je známa řada vylepšení tohoto základního schématu. Každý vyhledávací strom může být zredukován na „minimální vyhledávací strom" (Yu a kol., 1993) vynecháním nevětvících se uzlů neboli zkrácením cest. Pro jednoduchost tento mechanismus nezahrnujeme do detailních popisů algoritmů, byl však vzat v úvahu při experimentech. Za obecně nejpoužívanější variantu algoritmu lze doposud považovat tzv. „vylepšený" algoritmus větví a mezí, na který se budeme odkazovat podle původního názvu IBB (Improved Branch & Bound) (Fukunaga, 1990). Bez nároku na přesnost nazvěme *poklesem hodnoty kritéria* rozdíl mezi momentální hodnotou kritéria a hodnotou po odebrání jednoho příznaku. Nazvěme *špatným příznakem* takový příznak, jehož odebrání z pra-

covní množiny kandidátů způsobí pouze *mírný pokles* hodnoty kritéria. Obdobně označme za *dobrý příznak* takový, jehož odebrání způsobí *značný pokles* hodnoty kritéria. (V tuto chvíli není třeba vymezovat přesně pojmy mírný a značný.) Je zřejmé, že i při neměnné topologii výpočetního stromu lze definovat různá konkrétní přiřazení příznaků hranám stromu. Algoritmus IBB se snaží přiřadit *špatné* příznaky do pravé, tj. řidší části stromu, a *dobré* příznaky do levé, tedy hustší části stromu. Smysl tohoto řazení je v urychlení *růstu meze* v průběhu výpočtu. Nejprve jsou totiž v pravé části stromu zkoumány cílové podmnožiny získané odebíráním těch *špatných* příznaků z X_D, které způsobí co nejmenší pokles hodnoty kritéria. Rychlejší růst meze v počátečním stadiu výpočtu v kombinaci s umístěním *dobrých* příznaků v levé části stromu vede v pozdějších fázích výpočtu k častějšímu poklesu testovaných hodnot kritéria pod mez a tím k efektivnějšímu ořezání hustých větví.



Obr. 13.2 Příklad problému řešeného algoritmem větví a mezí, kde $d = 2$ optimální dvojice příznaků je vybírána z celkového počtu $D = 5$ příznaků tak, aby maximalizovala ilustrativní uměle definovanou kriteriální funkci.

Efekt tohoto heuristického *horizontálního řazení uzlů* je ilustrován na obr. 13.2. Ukázaný strom neodpovídá stromu „vylepšeného" algoritmu IBB protože první úroveň stromu není vhodně seřazena (sekvence příznaků 5, 3, 4 zvýrazněná šedivým pozadím). Změníme-li pořadí tak, aby odpovídalo IBB pořadí (5, 4, 3), algoritmus položí mez rovnu skutečnému optimu hned při prvním vyhodnocení listu a tím ušetří o jedno vyčíslení kritéria více, $J(4, 5)$. Všimněte si, že ve vyznačených vrcholech je vyhodnocování kritéria nadbytečné, poněvadž tyto vrcholy leží na cestě. Přeskočením těchto vrcholů zredukujeme výpočetní strom na již zmíněný „minimální vyhledávací strom". Algoritmus IBB v kombinaci s „minimálním vyhledávacím stromem" budeme v dalším výkladu považovat za referenční podobu algoritmu větví a mezí.

13.4.2 Nevýhody tradičních algoritmů větví a mezí

Každý algoritmus větví a mezí provádí řadu výpočtů, které mohou, ale také nemusí, vyústit v celkovou časovou úsporu a které by nebyly potřeba při využití algoritmu úplného prohledávání. Hodnota kritéria není počítána pouze pro cílové podmnožiny d příznaků $\overline{\chi}_{D-d}$, ale navíc také pro jejich nadmnožiny $\overline{\chi}_{D-d-j}$, $j = 1, 2,..., D - d$.

Princip algoritmu větví a mezí nezaručuje, že bude možné odříznout dostatek větví výpočetního stromu a že tedy celková doba výpočtu bude kratší než v případě algoritmu úplného prohledání. Skutečná rychlost výpočtu závisí vždy na vlastnostech použité krite-

riální funkce i na konkrétních zpracovávaných datech. Teoreticky nejhorší situaci můžeme ilustrovat nadefinováním umělé kriteriální funkce $J(\overline{\chi}_k) = |\overline{\chi}_k| \equiv D - k$ (viz obr. 13.6a). Tato funkce způsobí vyhodnocování v každém uzlu výpočetního stromu a vede tedy celkově ke zhruba dvojnásobnému počtu výpočtů, než by vyžadoval algoritmus úplného prohledávání. V případě úplného prohledávání je předem známo, kolik výpočtů bude provedeno, v případě algoritmu větví a mezí celkový počet výpočtů přesně predikovat nelze.

Nebezpečí příliš neefektivního běhu algoritmu větví a mezí vyplývá mimo jiné z těchto pozorování:

a) výpočty jsou tím pomalejší, čím blíže kořeni stromu jsou prováděny (zpracovávají se větší podmnožiny příznaků),

b) pravděpodobnost odříznutí větve je blíže kořeni naopak menší (hodnoty kritéria vnitřních uzlů jsou blíže kořeni větší, a tím spíše neklesnou pod dosavadní mez počítanou v listech).

Klasické algoritmy větví a mezí takto spotřebují většinu výpočetního času na pomalé vyhodnocování málo perspektivních vnitřních uzlů výpočetního stromu. Tento efekt je znatelný zejména pro $d \ll D$. Ve „vylepšeném" algoritmu IBB je navíc potřeba významné množství dodatečných výpočtů k heuristickému řazení uzlů stromu, jak uvidíme v následujícím textu.

13.4.3 Vylepšení „vylepšeného" algoritmu

Ukažme si nejprve zásadní nevýhodu „vylepšeného" algoritmu IBB z obrázku 13.2. Když algoritmus konstruuje první úroveň stromu, neboli když určuje pořadí následníků kořene, tak vyhodnocuje *pokles hodnoty kritéria* pro všech 5 momentálně použitelných příznaků z množiny doposud nepřiřazených příznaků Ψ. Pouze 3 příznaky jsou však poté použity, výsledky zbylých dvou výpočtů se dále nepoužijí.

Naším cílem je nalézt stejné (nebo nepříliš odlišné) řazení uzlů stromu jako v případě „vylepšeného" algoritmu IBB avšak s menším počtem vyhodnocování kritéria. K tomuto účelu použijeme jednoduchý predikční mechanismus. Další úrovně stromu budeme konstruovat ve dvou fázích. V první fázi pouze rychle odhadneme *poklesy hodnoty kritéria* pro všechny příznaky z Ψ. Příznaky seřadíme podle odhadnutých hodnot a pro konstrukci stromu použijeme pouze nutný počet příznaků q_k v pořadí od *největší hodnoty* poklesu. Pouze pro tyto vybrané příznaky poté dopočteme skutečné hodnoty kritéria a na jejich základě popřípadě poopravíme pořadí. Tímto způsobem lze snížit celkový počet vyhodnocování kritéria při konstrukci jednotlivých úrovní stromu na úroveň základního algoritmu větví a mezí, výsledný strom se však nebude příliš lišit od efektivně seřazeného stromu IBB.

K predikci používáme statistiku *poklesů hodnot kritéria* získanou v průběhu dosavadního výpočtu zvlášť pro každý jednotlivý příznak. Nazvěme $\boldsymbol{A} = [A_1, A_2, \ldots, A_D]^T$ *vektorem příspěvků* příznaků k hodnotě kritéria. Tento vektor bude pro každý příznak zaznamenávat průměrný pokles hodnoty kritéria způsobený odebíráním tohoto příznaku v různých okamžicích výpočtu. Počet vyhodnocování poklesu hodnoty kritéria pro jednotlivé příznaky budeme udržovat ve *vektoru čítačů* $\boldsymbol{B} = [B_1, B_2, \ldots, B_D]^T$. Pro formální popis tohoto algoritmu větví a mezí s částečnou predikcí (Branch & Bound with Partial Prediction BBPP) respektujeme konvence značení z knihy (Devijver a kol., 1982):

k – úroveň stromu ($k = 0$ značí kořen),

$\overline{\chi}_k = \{\xi_j \mid j = 1, 2, ..., D-k\}$ – momentální pracovní množina „kandidátských" příznaků v k-té úrovni stromu,

q_k – počet následníků zpracovávaného uzlu (v další úrovni stromu),

$Q_k = \{Q_{k,1}, Q_{k,2}, \ldots, Q_{k,q_k}\}$ – seřazená množina příznaků přiřazených hranám vedoucím k následníkům právě zpracovávaného uzlu (všimněte si, že „kandidátské" podmnožiny $\overline{\chi}_{k+1}$ jsou jednoznačně určeny pomocí příznaků $Q_{k,i}$, kde $i = 1, \ldots, q_k$),

$\boldsymbol{J}_k = [J_{k,1}, J_{k,2}, \ldots, J_{k,q_k}]^T$ – vektor hodnot kritéria odpovídajících následníkům právě zpracovávaného uzlu $J_{k,i} = J(\overline{\chi}_j \setminus \{Q_{k,i}\})$ pro $i = 1, ..., q_k)$,

$\Psi = \{\psi_j \mid j = 1, 2, \ldots, r\}$ – řídicí množina udržující r příznaků momentálně dostupných pro konstrukci další úrovně stromu, neboli pro sestavení množiny Q_k, množina Ψ tak zaručuje optimální topologii stromu,

$\mathscr{X}^* = \{x_j \mid j = 1, 2, \ldots, d\}$ – nejlepší doposud nalezená podmnožina d příznaků,

X^* – momentální mez (hodnota kritéria odpovídající množině $\mathscr{X}^*$).

Poznámka: Hodnoty q_j množiny Q_j a vektory $\boldsymbol{J}_j$ musí být vždy uloženy pro všechna $j = 0, \ldots, k$, aby byl umožněn zpětný průchod stromem (backtracking).

Kdykoli v průběhu výpočtu dojde v souvislosti s odebráním nějakého příznaku y_i z nějaké aktuální „kandidátské" podmnožiny $\overline{\chi}_k$ k výpočtu hodnoty kritéria $J(\overline{\chi}_k \setminus \{y_i\})$, bude následovat aktualizace *predikční informace*

$$A_{y_i} = \frac{A_{y_i} B_{y_i} + (J(\overline{\chi}_k) - J(\overline{\chi}_k \setminus \{y_i\})).1}{B_{y_i} + 1} \tag{13.3}$$

a

$$B_{y_i} = B_{y_i} + 1, \tag{13.4}$$

kde A_{y_i} i B_{y_i} jsou na začátku inicializovány hodnotou 0 pro všechna $i = 1, \ldots, D$.

Algoritmus větví a mezí s částečnou predikcí (BBPP)

Inicializace: $k = 0$, $\overline{\chi}_0 = X_D$, $\Psi = X_D$, $r = D$, X^* udává nejnižší strojově možnou hodnotu.

KROK 1: *Zvolme následníky právě zpracovávaného uzlu, a tím vytvořme další úroveň stromu.* Nejprve určeme jejich počet $q_k = r - (D - d - k - 1)$. Nyní sestavme uspořádanou množinu Q_k a vektor $\boldsymbol{J}_k$ takto: seřaďme všechny příznaky $\psi_j \in \Psi, j = 1, \ldots, r$, sestupně podle jim odpovídajících hodnot A_{ψ_j}, $j = 1, ..., r$, neboli

$$A_{\psi_{j_1}} \geq A_{\psi_{j_2}} \geq ... \geq A_{\psi_{j_r}},$$

a vyberme postupně prvních q_k z nich. Položme

$$Q_{k,i} = \text{ pro } i = 1, ..., q_k \quad \text{a} \quad J_{k,i} = J(\overline{\chi}_k \setminus \{\psi_{j_i}\}) \text{ pro } i = 1, ..., q_k.$$

Abychom zabránili budoucím redundantním vyhodnocováním kritéria, vylučme příznaky ψ_{j_1} z další konstrukce stromu, neboli položme $\Psi = \Psi \setminus Q_k$ a $r = r - q_k$.

KROK 2: *Otestujme nejpravějšího následníka (připojeného hranou Q_{k,q_k}).* Pokud $q_k = 0$, potom všichni následníci již byli otestováni a proto pokračujme Krokem 4 (backtracking). Pokud $J_{k,q_k} < X^*$ pokračujme Krokem 3. Jinak položme $\overline{\chi}_{k+1} = \overline{\chi}_k \setminus \{Q_{k,q_k}\}$. Pokud $k + 1 = D - d$, pak byl dosažen list, a proto pokračujme Krokem 5. Jinak pokračujme průchodem do další úrovně stromu: položme $k = k + 1$ a jděme na Krok 1.

KROK 3: *Následný uzel připojený hranou Q_{k,q_k} (včetně celého jeho podstromu) může být odříznut.* Vraťme příznak Q_{k,q_k} do množiny příznaků určených k další konstrukci stromu, neboli položme $\Psi = \Psi \cup \{Q_{k,q_k}\}$ a $r = r + 1$, $Q_k = Q_k \setminus \{Q_{k,q_k}\}$ a $q_k = q_k - 1$ a pokračujme jeho levým sousedem, jděme na Krok 2.

KROK 4: *Návrat do předchozí úrovně.* Položme $k = k - 1$. Pokud $k = -1$, pak celý strom byl již zkonstruován a algoritmus může být ukončen. Jinak vraťme příznak Q_{k,q_k} do množiny „kandidátů". Položme $\overline{\chi}_k = \overline{\chi}_{k+1} \cup \{Q_{k,q_k}\}$ a jděme na Krok 3.

KROK 5: *Aktualizujme hodnotu meze.* Položme $X^* = J_{k,q_k}$. Uložme doposud nejlepší nalezenou cílovou podmnožinu $\mathscr{X} = \overline{\chi}_{k+1}$ a jděme na Krok 2.

Obr. 13.3 Zjednodušený diagram algoritmu větví a mezí s částečnou predikcí (BBPP).

O vlastnostech algoritmu BBPP diskutujeme v odstavcích 13.4.5 a 13.4.6. Zjednodušený diagram algoritmu naleznete na obr. 13.3. Poznamenejme ještě, že formální popis BBPP se od IBB liší pouze v Kroku 1, kde by v případě IBB bylo místo prostého seřazení predikovaných hodnot $A_{\psi j}$ nutné před řazením nejprve spočítat všechny skutečné hodnoty $J(\overline{\chi}_k \setminus \{\psi_j\})$.

13.4.4 Rychlý algoritmus větví a mezí

Takzvaný rychlý algoritmus větví a mezí (Fast Branch & Bound, FBB) (Somol a kol., 2004) byl navržen s cílem ještě podstatněji zredukovat počet vyhodnocování kritéria ve vnitřních uzlech výpočetního stromu. K tomu účelu bylo dalekosáhleji využito predikčního mechanismu. Algoritmus se snaží využít znalost dosavadních poklesů hodnoty kritéria nejen k řazení uzlů, ale i při průchodu stromem do hloubky. Skutečné i predikované hodnoty jsou při konstrukci stromu využívány ekvivalentním způsobem s výjimkou dvou specifických situací:

1. predikce nemá smysl v úrovni listů a
2. testování predikované hodnoty oproti mezi nikdy nemůže vést samo o sobě k odříznutí podstromu.

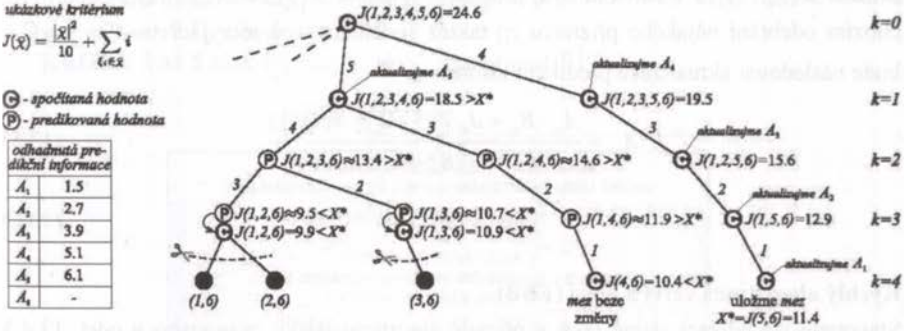
Zůstává-li v průběhu výpočtu predikovaná hodnota kritéria znatelně větší než *momentální mez*, je možné předpokládat, že ani skutečná hodnota nejspíše nepadá pod mez a odpovídající podstrom nelze odříznout. V takové situaci algoritmus jednoduše pokračuje konstrukcí další úrovně stromu. Přiblíží-li se ovšem v průběhu výpočtu predikovaná hodnota momentální mezi, popř. pod tuto mez klesne, možnost odříznutí podstromu je třeba ověřit dopočtením skutečné hodnoty kritéria. Pouze když se i skutečná hodnota ukáže nižší než mez, je podstrom algoritmem odříznut. Všimněme si, že tímto způsobem zůstává zaručena optimalita výsledného řešení, ačkoliv v řadě vnitřních uzlů stromu vůbec nedošlo k výpočtu skutečné hodnoty kritéria. Všimněme si dále, že ani případné nepřesnosti či chyby predikce optimalitu výsledku neovlivní, mohou pouze vést k průchodu částí stromu, která by za jiných okolností mohla být odříznuta. V sekci experimentů ukážeme, že ani v případě velmi výrazných chyb predikčního mechanismu neklesá efektivita algoritmu na úroveň algoritmů nevyužívajících predikce.

V rámci algoritmu FBB je nutné uchovávat informaci o tom, která hodnota kritéria v různých částech stromu byla získána tím kterým způsobem. K tomu využijeme *typového vektoru*, v němž pro každý uzel právě zpracovávané úrovně stromu bude zaznamenán symbol „P" nebo „C" podle toho, byla-li hodnota kritéria v uzlu predikována či skutečně vyčíslena. Informace o typu uzlu především rozlišuje, je-li případnou možnost odříznutí podstromu nutné nejdříve ověřit dopočtením hodnoty kritéria (tedy změnit typ z „P" na „C"), či je-li možné odříznutí rovnou provést (typ „C"). Informace z typových vektorů je dále využíváno při učení predikčního mechanismu. Záznamy ve *vektoru příspěvků* ***A*** a *vektoru čítačů* ***B*** jsou aktualizovány pouze v případě, že hodnota kritéria před odebráním i po odebrání příznaku je typu „C". Pokud bychom k aktualizaci predikční informace použili rozdílu predikovaných či smíšených hodnot kritéria, došlo by k zanesení chybné informace a funkčnost predikčního mechanismu by byla vážně narušena.

Obrázek 13.4 ilustruje popsaný mechanismus. Ukazuje průběh učení predikčního mechanismu, kdykoliv dojde ke skutečnému vyhodnocení dvou následných hodnot kritéria včetně následného využití naučené informace k predikci v podobě jednoduchého

odečtení. Predikované hodnoty jakožto pouhé aproximace hodnot skutečných nemohou vést k odříznutí podstromu a každá zdánlivá možnost odříznutí musí být proto ověřena spočtením hodnoty skutečné (uzly reprezentující množiny 1, 3, 6 a 1, 2, 6).

Výkonnost predikčního mechanismu se v průběhu algoritmu obvykle postupně zlepšuje. Na počátku, a to vzhledem k nedostatku informace, můžeme očekávat časté aktualizace vektoru příspěvků oproti ojedinělým predikcím. S postupem času a přibýváním „věrohodných" odhadů algoritmus využívá stále častěji predikce na úkor skutečných výpočtů, čímž naopak zpomaluje proces učení predikčního mechanismu. (V této souvislosti byl zaveden nepovinný parametr δ dovolující určit *minimálně nutný počet vyhodnocení* příspěvků příznaku, do jehož dosažení není predikce používána.)



A_1	1.5
A_2	2.7
A_3	3.9
A_4	5.1
A_5	6.1
A_6	-

Obr. 13.4 Ilustrace běhu predikčního mechanismu „rychlého algoritmu větví a mezí" na umělém problému výběru $d = 2$ z celkového počtu $D = 6$ příznaků.

Úkolem uvedeného predikčního mechanismu není pouze nahrazovat při průchodu výpočetním stromem výpočty predikcemi, ale také odhadnout okamžik (uzel), kdy již skutečná hodnota kritéria padla pod dosavadní mez a bylo by tedy možné zbytek větve odříznout. Především v tomto smyslu je třeba počítat s možností dvou typů chyb vyplývajících z principu predikce. Nazvěme *pesimistickou chybou predikce* situaci, když FBB zastaví predikci příliš brzy a začne místo ní vyhodnocovat skutečné hodnoty kritéria v důsledku mylného odhadu, že již lze podstrom odříznout. Označme naopak *optimistickou chybou predikce* situaci, pokud při průchodu stromem predikované hodnoty nenaznačují možnost odříznutí podstromu a algoritmus tedy pokračuje zbytečně průchodem do nižších úrovní. Chování algoritmu lze ovlivnit zavedením dodatečné *konstanty optimismu* γ takto: kdykoliv má dojít k predikci odečtením hodnoty zaznamenané ve vektoru příspěvků $\boldsymbol{A}$, nechť je odečtena tato hodnota vynásobená veličinou γ. Hodnoty $\gamma > 1$ takto způsobí rychlejší pokles hodnot kritéria a tím rychlejší přiblížení k dosavadní mezi, a tedy působí proti výskytu *optimistických* chyb predikce ve prospěch chyb *pesimistických*. Hodnoty $0 < \gamma < 1$ působí opačně, více viz odst. 13.4.5.1.

Pro formální popis rychlého algoritmu větví a mezí (Fast Branch & Bound, FBB) použijeme značení zavedené při popisu algoritmu BBPP (odst. 13.4.3) rozšířené o tyto konstanty a symboly:

$\delta \geq 1$ – *minimální nutný počet vyhodnocení* (standardně 1),

$\gamma \geq 0$ – *konstanta optimismu* (standardně 1),

$\boldsymbol{T}_k = [T_{k,1}, T_{k,2}, \ldots, T_{k,q_k}]^{\mathrm{T}}$, $T_{k,i} \in \{\mathsf{C}, \mathsf{P}\}$ pro $i = 1, \ldots, q_k$ – *typový vektor* (zaznamenává způsob získání hodnot $J_{k,i}$),

$\boldsymbol{V} = [v_1, v_2, \ldots, v_{q_k}]^{\mathrm{T}}$ – *pomocný řadicí vektor.*

Poznámka: Hodnoty q_j množiny Q_j a vektory $\boldsymbol{J}_j$, $\boldsymbol{T}_j$ musí být vždy uloženy pro všechna $j = 0, \ldots, k$, aby byl umožněn zpětný průchod stromem (backtracking).

Kdykoliv dojde v průběhu výpočtu v souvislosti s odebráním nějakého příznaku y_i z nějaké aktuální „kandidátské" podmnožiny $\overline{\chi}_k$ v k-té úrovni stromu k výpočtu hodnoty kritéria $J(\overline{\chi}_k \setminus \{y_i\})$ a zároveň byla předchozí hodnota $J(\overline{\chi}_k) \equiv J(\overline{\chi}_{k-1} \setminus \{y_i\})$ (po předchozím odebrání nějakého příznaku y_j) taktéž spočítána (což je vyjádřeno $T_{k-1,y_i} = \mathsf{C}$), bude následovat aktualizace predikční informace

$$A_{y_i} = \frac{A_{y_i} \cdot B_{y_i} + J_{k-1,y_i} - J(\overline{\chi}_k \setminus \{y_i\})}{B_{y_i} + 1}, \tag{13.5}$$

$$B_{y_i} = B_{y_i} + 1. \tag{13.6}$$

Rychlý algoritmus větví a mezí (FBB)

Stanovme inicializaci stejně jako v případě algoritmu BBPP popsaného v odst. 13.4.3 a navíc nastavme hodnoty δ a γ podle doporučení podaných v odst. 13.4.5.1.

KROK 1: *Zvolme následníky právě zpracovávaného uzlu, a tím vytvořme další úroveň stromu.* Nejprve určeme jejich počet $q_K = r - (D - d - k - 1)$. Nyní sestavme uspořádanou množinu Q_k a vektory $\boldsymbol{J}_k$ a $\boldsymbol{T}_k$ takto: pro všechny příznaky $\psi_j \in \Psi$, $j = 1, \ldots, r$, je-li $k + 1 < D - d$ (uzly nejsou listy) a $B_{\psi_j} > \delta$ (predikce povolena), položme

$$v_j = J_{k-1,q_{k-1}} - A_{\psi_j},$$

neboli předpovídáme odečtením predikční hodnoty odpovídající příznaku ψ_j od hodnoty kritéria získané v rodičovském uzlu. V jiných případech musí být hodnota kritéria spočtena

$$v_j = J(\overline{\chi}_k \setminus \{\psi_j\}).$$

Po získání všech hodnot v_j tyto hodnoty setřiďme vzestupně

$$v_{j_1} \le v_{j_2} \le \ldots \le v_{j_r}$$

a pro $i = 1, \ldots, q_k$ položme

$Q_{k,i} = \psi_{j_i}$,

$J_{k,i} = v_{j_i}$, pokud v_{j_i} je spočítaná hodnota, nebo

$J_{k,i} = J_{k-1,q_{k-1}} - \gamma . A_{\psi_{j_i}}$ v případě, že v_{j_i} je hodnota predikovaná,

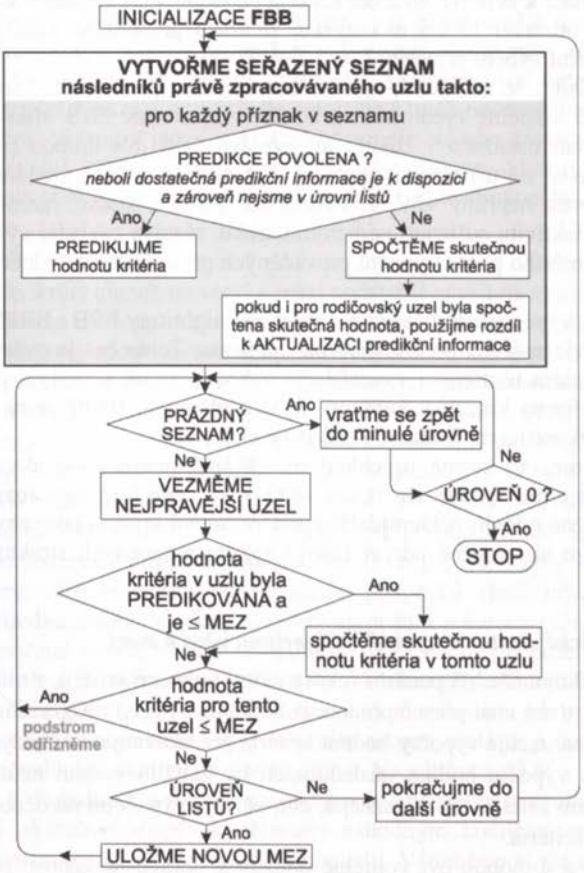
$T_{k,i}$ = C, pokud v_{j_i} je spočítaná hodnota, nebo

$T_{k,i}$ = P v případě, že v_{j_i} je hodnota predikovaná.

Položme $\Psi = \Psi \setminus Q_k$ a $r = r - q_k$ pro zamezení redundantního vyhodnocování kritéria.

KROK 2: *Otestujme nejpravějšího následníka (připojeného hranou Q_{k,q_k}).* Pokud q_k = 0, potom všichni následníci již byli otestováni, a proto pokračujme Krokem 4 (backtracking). Pokud T_{k,q_k} = P a $J_{k,q_k} < X^*$, spočtěme skutečnou hodnotu $J_{k,q_k} = J(\overline{\chi}_k \setminus \{Q_{k,q_k}\})$ a označme T_{k,q_k} = C. Pokud T_{k,q_k} = C a $J_{k,q_k} < X^*$, pokračujme Krokem 3. Jinak položme $\overline{\chi}_{k+1} = \overline{\chi}_k \setminus \{Q_{k,q_k}\}$. Pokud $k + 1 = D - d$, pak byl dosažen list a proto pokračujme Krokem 5. Jinak pokračujme průchodem do další úrovně stromu a položme $k = k + 1$a jděme na Krok 1.

KROKY 3 až 5 zůstávají stejné jako v algoritmu BBPP.



Obr. 13.5 Zjednodušený diagram tzv. rychlého algoritmu větví a mezí (FBB).

Poznámka: V Kroku 1 pro $k = 0$ výraz $J_{-1,q_{-1}}$ označuje hodnotu kritéria na množině všech příznaků $J(X_D)$. Zjednodušený diagram algoritmu naleznete na obr. 13.5.

13.4.5 Nové vlastnosti algoritmů využívajících predikce

Výkonnost kteréhokoliv algoritmu větví a mezí podstatně závisí na vlastnostech použité kriteriální funkce a zpracovávaných dat. Mimo tyto externí faktory, které ovlivňují vyhledávací proces ve všech variantách algoritmu (budou zkoumány podrobněji v odst. 13.4.6), vykazují nové predikční algoritmy BBPP a FBB několik specifických vlastností. Oba algoritmy využívají predikčního mechanismu založeného na heuristických předpokladech. Z toho důvodu nelze spoléhat na bezvadnou funkčnost prediktoru za všech okolností. Algoritmy BBPP i FBB budou nejefektivnější a nejvýrazněji překonají IBB, pokud jejich prediktor uspěje jak ve fázi učení, tak v pozdější fázi náhrady skutečných kriteriálních hodnot hodnověrnými odhady. Toto lze očekávat, zejména pokud individuální příspěvky příznaku k celkové hodnotě kritéria nebudou silně kolísat v kontextu různých podmnožin. V odstavci 13.4.6 ukazujeme, že tento předpoklad není příliš omezující alespoň v kontextu výběru příznaků.

Za předpokladu, že nedojde k naprostému výpadku predikčního mechanismu, bude algoritmus FBB znatelně rychlejší než BBPP. Algoritmus FBB obsahuje totiž nejen všechna vylepšení obsažená v BBPP, ale přidává další. Na druhou stranu výkonnost BBPP méně závisí na úspěšnosti predikce. Potenciální selhání predikce by v případě BBPP mělo pouze nepřímý vliv na výslednou rychlost, špatné řazení vnitřních uzlů stromu omezí efektivitu odříznutí podstromů, avšak zůstává základní výhoda oproti IBB v podobě vždy nižšího počtu výpočtů prováděných při konstrukci jednotlivých vnitřních úrovní stromu.

Oproti klasickým algoritmům větví a mezí oba algoritmy FBB i BBPP spotřebovávají určitý čas navíc správou predikčního mechanismu. Tento čas je ovšem zanedbatelný ve srovnání s časem ušetřeným vynecháním vyhodnocování kritéria zejména v případě nerekurzivních forem kritérií v kontextu výběru příznaků. BBPP je na rozdíl od FBB použitelný i s rekurzivními formami kritéria (viz obr. 13.7).

Poznamenejme, že kromě urychlení na základě predikce je aktuálně zkoumána i možnost prodeje prostoru za čas (Chen, 2003). Za cenu budování rozsáhlé paměťové struktury je možné odhalit některá další vyhodnocování kritéria jako zbytečná. V tomto přístupu je zatím ale obtížné udržet časově správu pomocných struktur v rozumných mezích.

13.4.5.1 Specifické vlastnosti rychlého algoritmu větví a mezí

Algoritmus FBB nemůže být použit s rekurzivními formami kritérií, v nichž je k výpočtu hodnoty $J(\overline{\chi}_k)$ třeba znát přesně předchozí hodnotu $J(\overline{\chi}_{k-1})$. Algoritmus FBB při průchodu větvemi nahrazuje výpočty hodnot kritéria predikovanými odhady, jejichž využití k rekurzivnímu výpočtu hodnot následujících by zatížilo systém neúnosnou chybou. Výhoda algoritmu FBB je tím výraznější, čím větší je výpočetní náročnost použité nerekurzivní formy kritéria.

Konstanty γ a δ mohou být volitelně použity k úpravě funkčnosti predikčního mechanismu. Za základní a v obecném případě nejvýhodnější nechť jsou považovány hodnoty $\gamma = 1$ a $\delta = 1$. Pro konkrétní problémy je možné, že odlišné hodnoty by výpočet

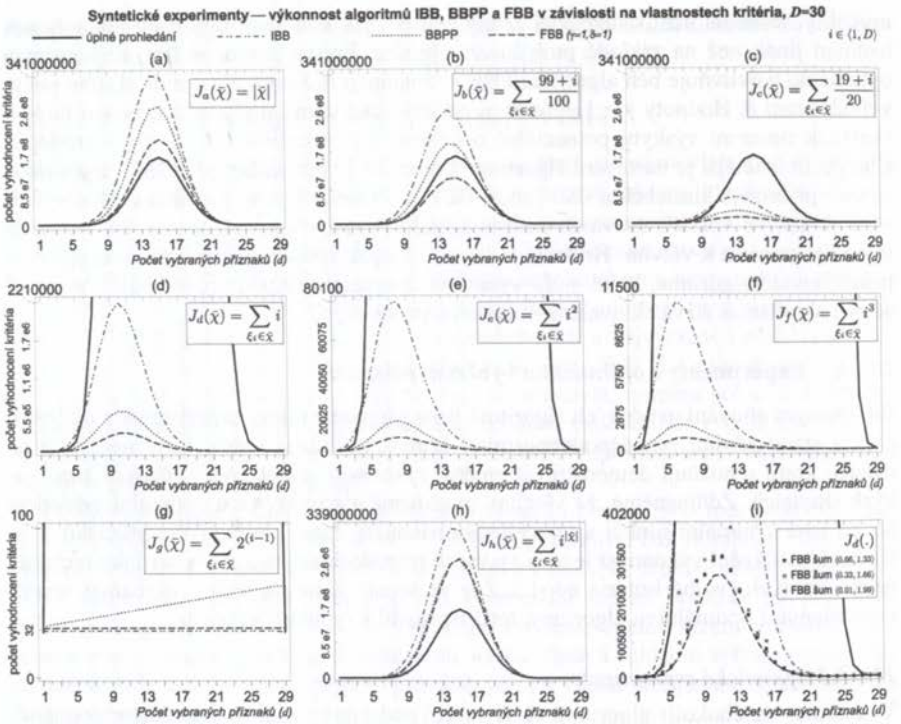
urychlily. Bohužel není k dispozici žádný způsob, jak konkrétní optimální hodnoty odhadnout jinak než na základě proběhlého výpočtu. Přesto je možné říci, že konstanta optimismu γ ovlivňuje běh algoritmu FBB významněji než konstanta minimálního počtu vyhodnocení δ. Hodnoty $\gamma > 1$ způsobují pesimistické chování algoritmu, neboli mohou sloužit k omezení výskytu potenciálně nebezpečnějších optimistických chyb predikce. Čím pesimističtější je nastavení algoritmu, tím méně je prováděno predikcí a algoritmus se více přibližuje klasickému IBB (popř. BBPP). Přesněji řečeno predikce budou využívány pouze ve vyšších úrovních stromu a budou zastavovány předčasně při průchodu stromem směrem k větvím. Hodnoty $0 < \gamma < 1$ naopak způsobují méně žádoucí optimistické chování algoritmu, které může vyústit ve zpomalení až pod úroveň IBB. Pro $\gamma = 0$ se pak algoritmus stává ekvivalentem úplného prohledání.

13.4.6 Experimenty s optimálním výběrem příznaků

Pro ilustraci chování uváděných algoritmů jsme připravili mimo experimenty s reálnými daty a standardními pravděpodobnostními kritérii též různé umělé testy nezávislé na datech, které umožňují demonstrovat změny výkonnosti algoritmů v různých modelových situacích. Zdůrazněme, že všechny uvažované algoritmy jsou optimální vzhledem ke zvolené kriteriální funkci a ve výsledku nacházejí tutéž podmnožinu příznaků. Výsledná klasifikační výkonnost je tedy ve všech případech stejná a závisí na faktorech zde nezkoumaných (volbě kritéria apod.). Zde se soustředíme na jediný významný rozdíl mezi různými optimálními algoritmy, totiž na rozdíl v rychlosti výpočtu.

13.4.6.1 Syntetické experimenty

Výkonnost kteréhokoliv algoritmu větví a mezí podstatně závisí na vlastnostech použitého kritéria a datového souboru. Abychom mohli ukázat vlastnosti algoritmů co nejobecněji, nadefinovali jsme sadu ilustrativních kriteriálních funkcí nezávislých na datech. Obrázek 13.6(a) ukazuje nejhorší možnou situaci, kdy je všem variantám algoritmu větví a mezí znemožněno odřezávat podstromy, tato situace je vyvolána prostřednictvím kriteriální funkce ohodnocující všechny příznaky jako ekvivalentní. Následující série obrázků (b) až (f) ukazuje, jak rostoucí vzájemná odlišnost příznaků (ve smyslu jejich příspěvku k hodnotě kritéria) vede k efektivnějšímu odřezávání podstromů. Je vidět, že algoritmy založené na predikci dokáží této rostoucí rozmanitosti příznaků využít lépe než klasický algoritmus IBB. Extrémní případ vidíme na obr. 13.6(g), kde každý příznak přispívá k hodnotě kritéria větší hodnotou, než je součet příspěvků všech příznaků s menším indexem. V důsledku dokáží všechny varianty algoritmu s *horizontálním řazením uzlů* konstruovat výpočetní strom, v němž je optimum nalezeno již při průchodu první větví a všechny zbylé větve jsou ihned poté odříznuty. V tomto případě dojde k nalezení optima v lineárním čase. Obrázek 13.6(h) ukazuje, jakým způsobem lze vyřadit z činnosti predikční mechanismus algoritmu FBB. Takováto situace nastává, je-li hodnota kritéria silně závislá na velikosti zkoumané podmnožiny. Na obrázku 13.6(i) vidíme klesající efektivitu FBB v důsledku vlivu šumu na učení – zde násobíme v průběhu učení každý zaznamenávaný skutečný příspěvek příznaku náhodným koeficientem generovaným podle rovnoměrného rozložení z vhodného intervalu. Všimněme si, že ani poměrně výrazné zašumění (interval [0,01; 1,99]) nesníží celkovou efektivitu algoritmu FBB na úroveň IBB.



Obr. 13.6 Umělá kritéria demonstrující chování algoritmu větví a mezí. (a) Nejhorší možný scénář zabraňující zcela odříznutím podstromů. (b) až (f) Série příkladů ukazující efekt zlepšování výkonnosti se vzrůstajícími rozdíly mezi příznaky. (g) Nejlepší možný scénář vedoucí k nejefektivnějšímu možnému odřezávání podstromů. (h) Scénář selhání učení v algoritmu FBB. (i) Vliv šumu na učení v algoritmu FBB.

Z předchozích pozorování lze vyvodit, že výkonnost algoritmů větví a mezí závisí především na následujících charakteristikách procesu vyhodnocování kritéria:

1. Všechny příznaky by mezi sebou měly vykazovat stabilní a měřitelné odlišnosti ve smyslu příspěvku k hodnotě kritéria vyhodnocovaného na libovolných podmnožinách. Čím podobnější jsou příspěvky jednotlivých příznaků, tím horší výkonnost kterékoliv algoritmu větví a mezí lze očekávat. Příčina je prostá – prochází-li algoritmus od kořene k listům, aktuální hodnota kritéria klesá o příspěvky jednotlivých odebíraných příznaků. V případě velmi podobných příspěvků u všech příznaků bude aktuální hodnota kritéria velmi podobná v na libovolném uzlu na stejné úrovni stromu. Je tedy méně pravděpodobné, že v některém uzlu dané úrovně časem klesne pod mez a umožní tak odříznutí podstromu.

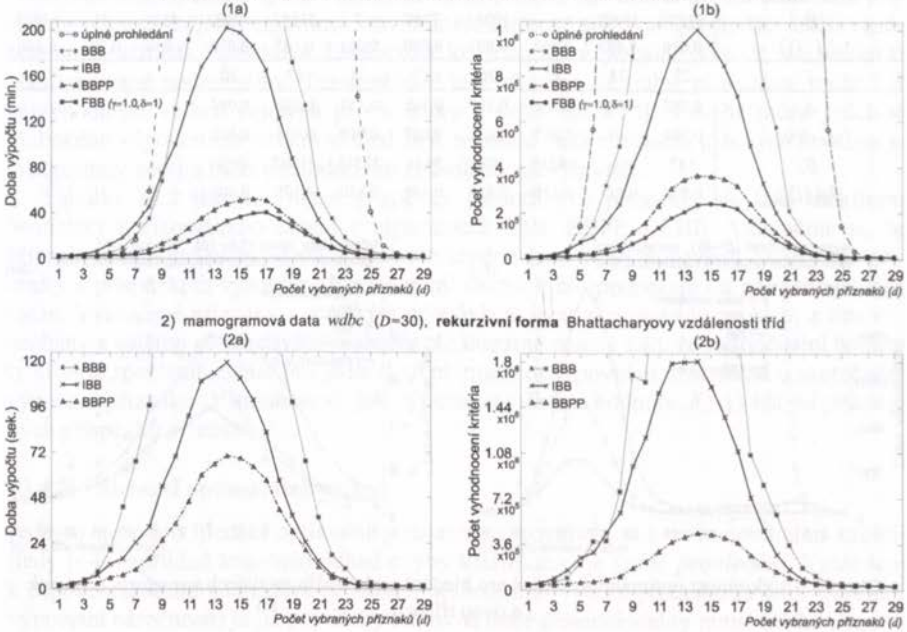
2. Příspěvek příznaku by neměl příliš kolísat v kontextu různých podmnožin. Nestabilní či zašuměné příspěvky příznaku zhoršují učící schopnost predikčního mechanismu (s důsledky hlavně v FBB). Navíc heuristika horizontálního řazení v takovém případě produkuje více či méně náhodná pořadí uzlů stromu, a tím podstatně sníží efektivitu odřezávání podstromů ve všech algoritmech IBB, BBPP a FBB (viz obr. 13.6(i)).

3. Hodnota kritéria by neměla záviset příliš silně na velikosti vyhodnocované podmnožiny (neboli na momentální úrovni stromu). Příliš vysoké rozdíly mezi příspěvkem téhož příznaku ve vztahu k různě velkým podmnožinám mohou opět zhoršit funkčnost predikčního mechanismu v FBB. Heuristika horizontálního řazení algoritmů IBB, BBPP a FBB také v tomto případě produkuje nevýhodná pořadí uzlů (viz obr. 13.6(h)).

Celkově lze pozorovat, že algoritmy využívající predikce (BBPP, FBB) vykazují větší efektivitu než referenční IBB pro všechna testovaná syntetická kritéria, ať už šlo o kritéria uměle komplikující či urychlující hledací proces.

13.4.6.2 Experimenty na reálných datech

Algoritmy zde testujeme na mamogramových datech *wdbc* o dvou třídách z Wisconsin Diagnostic Breast Centra (30 příznaků, 357 záznamů o zdravých a 212 nemocných pacientech) a datech *waveform* o třech třídách (40 příznaků, 1691 vzorků 1. třídy a 1693 vzorků třídy 2., třetí třída je vynechána vzhledem k omezení kriteriální funkce). Oba datové soubory pocházejí z databáze UCI (Asuncion a kol., 2007). Dále jsme použili řečová data *speech* o dvou třídách pocházející z British Telecomu (15 příznaků, 682 záznamů slova „yes" a 736 záznamů slova „no") získaná z CVSSP University of Surrey. Abychom zdůraznili univerzálnost zkoumaných algoritmů, provedli jsme testy se třemi různými kriteriálními funkcemi: divergencí, Bhattacharyovou vzdáleností a Patrickovou–Fischerovou vzdáleností (Devijver a kol., 1982). Použili jsme jak rekurzivní (bylo-li

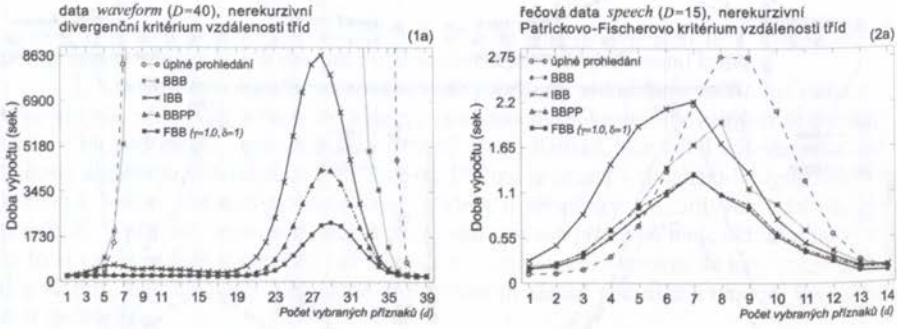


Obr. 13.7. Výkonnost optimálních metod pro hledání podmnožin na mamogramových datech *wdbc* o 2 třídách.

to možné), tak nerekurzivní formu Bhattacharyovy vzdálenosti. Výkonnost různých metod je ilustrována na obrázcích 13.7 a 13.8, které obsahují (a) graf výpočetního času a v případě dat *wdbc* též (b) graf počtu skutečně vyhodnocených hodnot kritéria. Tabulka 13.2 ukazuje statistiky související s učením predikčního mechanismu v algoritmech BBPP a FBB (k výpočtu statistik byl použit algoritmus IBB, který vyhodnocuje více hodnot a poskytuje tak informativnější výsledky). Tabulka ukazuje vektory ***A*** (průměrné příspěvky příznaků), ***B*** (počty vyhodnocení příspěvku příznaků), odchylku vyhodnocovaných hodnot A_i a individuální hodnoty kritéria $J(\{i\})$ spočtené pro jednotlivé příznaky zvlášť. Při použití nerekurzivních kriteriálních forem jsme nastavili všechny algoritmy tak, aby konstruovaly tzv. strom minimálního řešení (Yu a kol., 1993), což není možné v případě forem rekurzivních. Parametry FBB byly nastaveny na hodnoty $\gamma = 1$ a $\delta = 1$.

Tabulka 13.2 Statistiky příspěvků příznaků k hodnotě kritéria pro datový soubor *wdbc*. Spočteno algoritmem IBB při výběru 15 z 30 příznaků za použití Bhattacharyovy vzdálenosti. Nalezená podmnožina: $X = \{1, 3, 4, 6, 7, 11, 14, 15, 16, 17, 21, 23, 24, 26, 27\}$; $J_B(X) = 5{,}741$; $J_B(X_D) = 7{,}53989$

i	1	2	3	4	5	6	7	8	9	10	11
A_i	0,269	0,066	0,272	0,651	0,118	0,114	0,195	0,130	0,085	0,077	0,214
A_i *st. odch.*	0,076	0,022	0,098	0,048	0,030	0,020	0,041	0,029	0,028	0,019	0,087
B_i	312	95630	525	4	30327	56149	1400	23549	93275	89565	395
$J_B(\{i\})$	0,586	0,113	0,629	0,641	0,073	0,307	0,498	0,790	0,064	0,003	0,463
i	12	13	14	15	16	17	18	19	20	21	22
A_i	0,119	0,126	0,843	0,127	0,133	0,441	0,122	0,105	0,198	0,340	0,083
A_i *st. odch.*	0,021	0,040	0	0,034	0,031	0,004	0,026	0,015	0,051	0,108	0,026
B_i	51923	12972	1	10083	7746	7	31892	77102	452	31	91836
$J_B(\{i\})$	0,010	0,483	0,803	0,003	0,050	0,081	0,103	0,032	0,036	0,787	0,140
i	23	24	25	26	27	28	29	30			
A_i	0,267	0,797	0,095	0,199	0,103	0,132	0,123	0,094			
A_i *dev.*	0,084	0,011	0,017	0,031	0,027	0,019	0,015	0,032			
B_i	142	2	88359	1037	9441	32543	51707	38630			
$J_B(\{i\})$	0,816	0,811	0,110	0,327	0,389	0,820	0,175	0,101			



Obr. 13.8 Výkonnost optimálních metod pro hledání podmnožin na datech *waveform* a *speech* o dvou třídách.

Exponenciální charakter problému optimálního vyhledávání je dobře ilustrován faktem, že pro 15dimenzionální data je ještě úplné prohledání srovnatelně rychlé s algo-

ritmy větví a mezí, zatímco pro data 30dimenzionální při hledání podmnožiny $d = 15$ (obrázek 13.7) již úplné prohledání potřebuje vyhodnotit přibližně 140krát více hodnot kritéria než referenční algoritmus větví a mezí IBB. U 40dimenzionálních dat při hledání podmnožiny $d = 20$ (obrázek 13.8) je tento rozdíl již $1{,}3.10^8$násobný.

Podobně jako v případě předchozích umělých testů všechny výsledky (obrázky 13.7 a 13.8) potvrzují, že oba algoritmy využívající predikce (BBPP a FBB) nacházejí řešení nejrychleji, v podstatě vždy překonávají referenční IBB. V závislosti na datovém souboru a použitém kritériu algoritmus FBB prokazuje schopnost řešit úlohu výběru příznaků 1,5× až 10× rychleji než IBB, zejména při výpočetně nejnáročnější volbě d blízko $D/2$.

Příklad *wdbc* (obrázek 13.7) ukazuje očekávatelné chování zkoumaných algoritmů, v tomto případě s využitím Bhattacharyovy vzdálenosti jakožto kritéria. Tabulka 13.2 v tomto případě popisuje velmi přesně naučenou predikční informaci (jak potvrzuje nízká odchylka A_i). Rozdíly mezi jednotlivými příznaky jsou viditelné a stabilní. V souladu s tím heuristika horizontálního řazení i predikční mechanismy úspěšně splnily svou funkci.

Příklad *waveform* (obrázek 13.8) ukazuje zajímavý fenomén, kdy všechny algoritmy s horizontálním řazením IBB, BBPP a FBB běžely překvapivě rychle až do velikosti hledané podmnožiny $d = 20$. Nabízí se vysvětlení, že zhruba polovina příznaků tohoto datového souboru reprezentuje pouze neinformativní šum (potvrzeno dokumentací souboru). Rozdíl mezi odhadnutými příspěvky informativních a šumových příznaků zde byl výborně využit heuristikou horizontálního řazení uzlů stromu.

Příklad řečových dat *speech* (obrázek 13.8) představuje úlohu, při níž pokročilé mechanismy modernějších algoritmů větví a mezí částečně selhávají především kvůli vlastnostem použitého Patrickova–Fischerova kritéria, které je silně závislé na velikosti ohodnocované podmnožiny. Tento příklad také ukazuje, že úplné prohledání může být nejvýhodnější volbou zejména pro hodnoty d velmi blízké 0, u nichž může průchod hlubokého výpočetního stromu vzhledem k relativně malému počtu listových konfigurací algoritmy větví a mezí rozhodujícím způsobem znevýhodnit.

Tabulka 13.2 ilustruje nejen příspěvky jednotlivých příznaků, ale také efektivitu heuristiky horizontálního řazení v algoritmech IBB, BBPP a FBB. Všimněme si, že příznaky s nízkým příspěvkem k hodnotě kritéria jsou vyhodnocovány častěji než příznaky s příspěvkem vysokým. Horizontální řazení takto podporuje co nejrychlejší růst meze. Významné příznaky s vysokým příspěvkem jsou akceptovány rychleji, a tím vynechány z dalšího prohledávání. Tabulka též ilustruje známý fakt, že individuální hodnoty kritéria spočítané zvlášť pro jednotlivé příznaky nevypovídají dostatečně o skutečném významu příznaku. Všimněme si, jak výrazně se některé hodnoty $J(\{i\})$ liší od průměrných příspěvků příznaků A_i.

13.4.7 Shrnutí optimálních metod

Jedinou metodou hledání optimální podmnožiny použitelnou s nemonotónními kritérii (tedy je-li například kritériem odhad chyby klasifikace) je *úplné prohledání*. Vzhledem k exponenciálnímu charakteru problému optimálního výběru příznaků (a tedy extrémní výpočetní náročnosti) je již pro úlohy relativně malé dimenzionality nutné hledat alternativní metody i za cenu omezení se na monotónní kritéria. Řada nedávných vylepšení *algoritmu větví a mezí*, především v podobě predikčních algoritmů FBB a BBPP, vyústi-

la do deseti- až stonásobného zrychlení oproti základní variantě algoritmu (v závislosti na konkrétním datovém souboru a kritériu).

Je ovšem potřeba zdůraznit, že i přes uvedený pokrok zůstávají optimální metody v principu exponenciální a tedy velmi pomalé. Není-li pro jejich aplikaci zvláštní důvod, bývá praktičtější využít některou z metod suboptimálních. Suboptimální metody jsou obvykle flexibilnější a schopné řešit problémy podstatně větších dimenzionalit, přičemž nalézaná řešení nemusí být v konečném důsledku nutně horší než řešení optimální. Volba vhodného kritéria má větší význam než časem draze zaplacené nalezení optima vzhledem k nevhodně zvolenému kritériu. Pro podrobnější diskusi optimálních metod odkazujeme na článek (Somol a kol., 2004).

13.5 Suboptimální vyhledávací metody

Přes pokroky v oblasti optimálních metod výběru příznaků (Somol a kol., 2004), (Nakariyakul a kol., 2007) je u většiny reálných úloh v kontextu statistického rozpoznávání obvykle nutné se obracet k suboptimálním metodám.

Známe velké množství nejrůznějších suboptimálních metod založených na nejrůznějších principech a poskytujících různou míru kompromisu mezi efektivitou a kvalitou nalezeného řešení. Velké množství metod je založeno na sledování největšího gradientu v prostoru přípustných řešení – nejtypičtějšími metodami jsou metody sekvenčního vyhledávání (Devijver a kol., 1982), metody Plus-*l*-Minus-*r* (Devijver a kol., 1982), jejich zobecněné verze (Devijver a kol., 1982), plovoucí vyhledávání (Pudil a kol., 1994a), (Somol a kol., 1999), (Nakariyakul a kol., 2009), oscilační vyhledávání (Somol a kol., 2000), (Somol a kol., 2008b) atd. Deterministické metody však mohou být náchylné k uvíznutí v lokálním extrému, proto je velmi často v nějaké formě využívána randomizace vyhledávacího procesu – hlavními zástupci těchto přístupů jsou metody Las Vegas, metoda Relief (Kononenko, 1994), (Sun, 2007), metoda Focus (Arauzo-Azofra a kol., 2003), simulované žíhání, např. (Debuse a kol., 1997), evoluční (genetické) algoritmy (Hussein a kol., 2001), vyhledávání tabu (Zhang a kol., 2002), metody mravenčí kolonie, např. (Jensen, 2006), randomizované oscilační vyhledávání (Somol a kol., 2000), ale také metoda řazení respektující závislost příznaku DAF (Somol a kol., 2011).

Pro výpočetně nejnáročnější problémy velmi velké dimenzionalty jsou obvykle zanedbávány vztahy mezi příznaky (metoda nejlepších individuálních příznaků, Best Individual Features, BIF, viz např. (Yang a kol., 1997)), popř. jsou přijímána nejrůznější zjednodušení či omezení rozsahu prohledávání. Řada metod umožňuje omezit prostor prohledávání uživatelskými parametry (Pudil a kol., 1994a) (Somol a kol., 2000) a (Somol a kol., 2008b), popř. se předpokládá, že uživatel pomocí parametrů rozsah prohledávání přesně specifikuje (Hussein a kol., 2001) a (Zhang a kol., 2002).

Ještě významnější než volba metody a jejích parametrů se ukazuje volba kriteriální funkce (viz filtry vs. pouzdra, odst. 13.3.2). Při nevhodné volbě kriteriální funkce nelze ani v případě nalezení optima očekávat optimální přesnost výsledného klasifikátoru. Tomuto a dalším problémům ovlivňujícím výsledek výběru příznaků se věnujeme též v oddílu 13.9.

Integrální součástí každého procesu výběru příznaků je rozhodnutí o počtu příznaků, které máme vybrat. Určení správné dimenzionality podprostoru vybraných příznaků je

obtížný problém, který je v obecnosti nad rámec této kapitoly. V další diskusi budeme nicméně rozlišovat dva typy metod FS: tzv. metody *d*-parametrizované a *d*-optimalizující. Většina existujících metod je *d*-parametrizovaných, tj. vyžadujících od uživatele rozhodnutí o kardinalitě výsledné podmnožiny příznaků. V oddílu 13.6 bude popsána *d*-optimalizující deterministická procedura, která optimalizuje současně jak velikost podmnožiny, tak její složení. Mezi nedeterministické *d*-optimalizující metody patří zejména evoluční algoritmy (Caruana a kol., 1994), (Chaikla a kol., 1999), (Mayer a kol., 2000) a (Hussein a kol., 2001).

V následujícím textu uvádíme základní přehled několika hlavních nástrojů užitečných pro problémy různé složitosti, založených většinou na konceptu sekvenčního vyhledávání (viz odst. 13.5.2).

13.5.1 Individuálně nejlepší příznaky

Metoda *individuálně nejlepších příznaků* (Best Individual Features, BIF) je nejrychlejší metodou výběru příznaků. Každý příznak je nejprve individuálně ohodnocen za použití zvoleného kritéria. Podmnožiny jsou pak vybrány prostě volbou prvních *d* individuálně nejlepších příznaků. Tento přístup je nejjednodušší, ale zároveň potenciálně nejslabší z hlediska optimalizační schopnosti – při výběru totiž zcela ignoruje možné komplikované závislosti mezi příznaky. Pro řešení problému FS při velké počáteční dimenzionalitě je to však často jediný aplikovatelný přístup. Metoda BIF je takto standardem pro kategorizaci textových dokumentů (Yang a kol., 1997), (Sebastiani, 2002), vyhledávání genů (Xing, 2003) (Saeys a kol., 2007) a v dalších aplikacích, kde dimenzionalita problému roste do tisíců.

Metoda BIF může být ovšem výhodná i v jiných situacích, protože se uplatňuje úspěšně tam, kde pokročilejší metody selhávají a produkují nestabilní výsledky, popř. vedou k přeučení a ztrátě schopnosti zobecnění (viz oddíl 13.9). Tyto situace nastávají nejen při problémech velmi velké dimenzionality, ale složitější metody jsou obecně tím náchylnější k selhání, čím větší je nepoměr mezi příliš velkou dimenzionalitou vůči příliš malému počtu vzorků trénovacího souboru. Není-li však úloha zatížena zmíněnými omezujícími okolnostmi, pokročilejší metody výběru příznaků mají větší potenciál dosáhnout lepších výsledků.

13.5.2 Sekvenční vyhledávání

Pro zjednodušení další diskuse se zaměříme na sekvenční vyhledávací metody. Označme Y množinu všech příznaků ($|Y| = D$). Většina známých sekvenčních algoritmů výběru příznaků sdílí stejný základní mechanismus přidávání a odebírání příznaků do stávající pracovní podmnožiny a z ní pryč. Příslušné kroky lze popsat uvedenými definicemi (pro zjednodušení budeme uvažovat pouze nezobecněné algoritmy (Devijver a kol., 1982), které zpracovávají v daném okamžiku pouze jeden příznak).

Definice 13.1. Pro stávající množinu příznaků X_d nechť f^+ je příznak takový, že

$$f^+ = \arg\max_{f \in Y \setminus X_d} J^+(X_d, f), \tag{13.7}$$

kde $J^+(X_d, f)$ označuje kriteriální funkci použitou pro vyhodnocení podmnožiny získané přidáním f do X_d, kde $f \in Y \setminus X_d$. Pak budeme říkat, že $ADD(X_d)$ je operace přidání příznaku f^+ do X_d za účelem vytvoření množiny X_{d+1}, jestliže

$$ADD(X_d) \equiv X_d \cup \{f^+\} = X_{d+1}, \quad X_d, X_{d+1} \subset Y. \tag{13.8}$$

Definice 13.2. Nechť f^- je ve stávající množině příznaků X_d takový příznak, že

$$f^- = \arg\max_{f \in X_d} J^-(X_d, f), \tag{13.9}$$

kde $J^-(X_d f)$ označuje kriteriální funkci použitou k vyhodnocení podmnožiny získané vyloučením f z X_d, je-li $f \in X_d$. Pak budeme říkat, že $REM(X_d)$ je operace vyloučení příznaku f^- z množiny X_d za účelem vytvoření množiny X_{d-1}, jestliže

$$REM(X_d) \equiv X_d \setminus \{f^-\} = X_{d-1}, \quad X_d, X_{d-1} \subset Y. \tag{13.10}$$

S cílem zjednodušit označení pro opakující se použití operací FS zavedeme tato užitečná označení:

$$X_{d+2} = ADD(X_{d+1}) = ADD(ADD(X_d)) = ADD^2(X_d),$$
$$X_{d-2} = REM(REM(X_d)) = REM^2(X_d) \tag{13.11}$$

a obecněji

$$X_{d+\delta} = ADD^\delta(X_d), \quad X_{d+\delta} = ADD^\delta(X_d). \tag{13.12}$$

Poznamenejme, že ve standardních sekvenčních metodách výběru příznaků $J^+(.)$ a $J^-(.)$ označují

$$J^+(X_d, f) = J(X_d \cup \{f\}), \; J^-(X_d, f) = J(X_d \setminus \{f\}), \tag{13.13}$$

kde $J(.)$ je kriteriální funkce založená obvykle buď na některé vhodné pravděpodobnostní míře, nebo na odhadu úspěšnosti klasifikace (viz filtry a pouzdra, odst. 13.3.2 (Kohavi a kol. 1997)).

13.5.3 Nejjednodušší sekvenční výběr

Nejjednodušší a přitom široce používanou variantou sekvenčního vyhledávání jsou metody *sekvenčního dopředného výběru* (SFS) a *sekvenčního zpětného výběru* (SBS) (Whitney, 1971) (Devijver a kol., 1982). Metoda SFS iterativně přidává (SBS odebírá) vždy v daném kroku jeden příznak tak, aby byla maximalizována okamžitá hodnota kritéria tak dlouho, až je dosaženo požadované velikosti cílové podmnožiny příznaků. Metoda SFS začíná od prázdné množiny a je proto známa jakožto metoda typu *bottom-up*. Metoda SBS začíná od množiny všech příznaků a je proto známa jako metoda typu *top-down*.

Sekvenční dopředný výběr (SFS)

Algoritmus generuje podmnožinu d příznaků zdola: $X_d = ADD^d(\emptyset)$.

Sekvenční zpětný výběr (SBS)

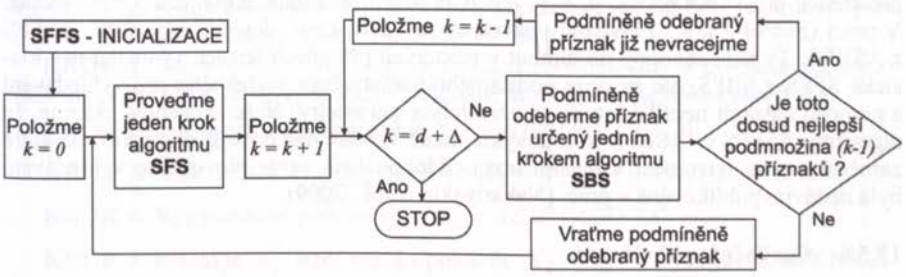
Algoritmus generuje podmnožinu d příznaků shora: $X_d = REM^{|Y|-d}(Y)$.

SBS jakožto metoda typu top-down od počátku výpočtu vyhodnocuje vztahy mezi všemi příznaky. Dává tak teoreticky šanci odhalit složitější vztahy mezi nimi a lépe tak rozhodovat o důležitosti jednotlivých příznaků. Metoda SBS je z téhož důvodu ale také výpočetně náročnější a oproti SFS více náchylná k numerickým problémům. Metoda SFS je z těchto důvodů často upřednostňována.

Jako většina dřívějších sekvenčních metod jak SFS, tak SBS trpí takzvaným hnízděním podmnožin příznaků (rozhodnutí o přidání, popř. odebrání, příznaku není v pozdější fázi výpočtu nikdy revokováno), které významně zhoršuje schopnost dosáhnout optima. Prvním pokusem o překonání tohoto problému bylo použití algoritmu Plus-*l*- Take away-*r* (také známého jako (*l*, *r*)) nebo zobecněného algoritmu (*l*, *r*) (Devijver a kol., 1982), které oba zahrnují postupný proces přidávání i odebírání příznaků. Tatáž základní myšlenka ovšem v podstatně rozšířené a složitější formě představuje základ plovoucího vyhledávání.

13.5.4 Sekvenční plovoucí vyhledávání

Procedura *sekvenčního dopředného plovoucího vyhledávání* (Sequential Forward Floating Search, SFFS) (Pudil a kol., 1994a) spočívá v tom, že po každém dopředném kroku aplikujeme zpětné kroky tak dlouho, dokud jsou výsledné podmnožiny lepší než ty, které byly předtím vyhodnoceny jako nejlepší na dané úrovni (pro danou dimenzionalitu). Z toho plyne, že v případě, kdy přechodný okamžitý výsledek nezlepšuje předchozí výsledek pro danou dimenzionalitu, k žádným zpětným krokům vůbec nedojde. Totéž platí pro dopředné kroky ve zpětné verzi procedury – *sekvenční zpětné plovoucí vyhledávání* (Sequential Backward Floating Search, SBFS). Oba algoritmy dovolují samořízené vratné kroky, které umožňují dosáhnout kvalitního řešení dynamicky flexibilním přepínáním mezi kroky dopřednými a zpětnými. Lze říci, že oba algoritmy provádějí pouze výpočty, které jsou potřeba k vylepšení doposud známých řešení, aniž by bylo nutné nastavovat jakékoliv uživatelské parametry.



Obr. 13.9 Algoritmus sekvenčního dopředného plovoucího výběru.

Sekvenční dopředný plovoucí výběr (SFFS)

Algoritmus generuje podmnožinu d příznaků zdola s volitelným, rozsah prohledávání omezujícím parametrem $\Delta \in [0, D-d]$ (pro hledání v plném rozsahu položme $\Delta = D-d$).

KROK 1: Položme $X_0 = \emptyset$, $k = 0$.

KROK 2: Přidejme k pracovní množině relativně nejlepší příznak pomocí operace $X_{k+1} = ADD(X_k)$, $k = k + 1$.

KROK 3: Opakujme operaci odebrání relativně nejhoršího příznaku pomocí operace $X_{k-1} = REM(X_k)$, $k = k - 1$, tak dlouho, dokud tímto způsobem nacházíme pro menší k lepší než doposud známá řešení.

KROK 4: Pokud $k < d + \Delta$, jděme na Krok 2.

Podrobný formální popis této dnes již klasické procedury lze nalézt v článku (Pudil a kol., 1994a). Nicméně základní myšlenka je poměrně jednoduchá a může být dostatečně ilustrována obrázkem 13.9. (Podmínka $k = d + \Delta$ ukončuje algoritmus poté, co byla nejen dosažena cílová podmnožina d příznaků, ale popřípadě i vylepšena pomocí zpětných vratných kroků z dimenzionalit větších než d.)

Plovoucí vyhledávací algoritmy lze považovat za univerzální nástroje, které nejen překonávají všechny své předchůdce, ale které si přitom uchovávají přednosti, jež nemají pozdější sofistikovanější algoritmy. Mají schopnost nalézat dobrá řešení pro všechny dimenze daného problému jedním průchodem algoritmu. Celková rychlost vyhledávání je dostatečně vysoká pro většinu praktických problémů.

Sekvenční zpětný plovoucí výběr (SBFS)

Algoritmus generuje podmnožinu d příznaků shora s volitelným, rozsah prohledávání omezujícím parametrem $\Delta \in [0, D - 1]$ (pro hledání v plném rozsahu položme $\Delta = D - 1$).

KROK 1: Položme $X_{|Y|} = Y$, $k = |Y|$.

KROK 2: Odeberme z pracovní množiny relativně nejhorší příznak pomocí operace $X_{k-1} = REM(X_k)$, $k = k - 1$.

KROK 3: Opakujme operaci přidání relativně nejlepšího příznaku pomocí operace $X_{k+1} = ADD(X_k)$, $k = k + 1$, tak dlouho, dokud tímto způsobem nacházíme pro větší k lepší než doposud známá řešení.

KROK 4: Pokud $k > d + \Delta$, jděme na Krok 2.

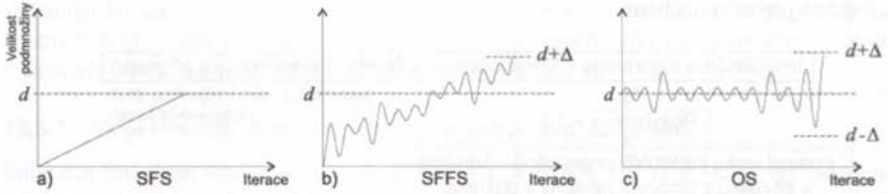
13.5.5 Další rozvoj principu plovoucího vyhledávání

Poté, co se plovoucí algoritmy úspěšně prosadily jako obecně přijímaný univerzální prostředek pro výběr příznaků, byla jejich základní idea dále zkoumána a rozšiřována. V práci (Somol a kol., 1999) byly navrženy tzv. *adaptivní plovoucí algoritmy*, ASFFS a ASBFS. Ty jsou schopny dosáhnout v některých případech lepších výsledků než klasické SFFS a SBFS, ale za cenu podstatného nárůstu času potřebného pro vyhledávání a nutnosti nastavit nepříliš intuitivní uživatelské parametry. Naše zkušenost ukazuje, že algoritmy ASFFS či ASBFS jsou obvykle méně vhodné než novější algoritmy, na které zaměříme svou pozornost v dalším textu. Zdokonalená verze plovoucího vyhledávání byla nedávno publikována v práci (Nakariyakul a kol., 2009).

13.5.6 Oscilační vyhledávání

Novější, tzv. *oscilační vyhledávání* (Oscillating Search, OS) (Somol a kol., 2000) může být považováno za metaproceduru, která využívá jiné metody výběru příznaků jako podprocedury ve svém vlastním vyhledávání. Celý koncept oscilačního vyhledávání je velmi flexibilní a umožňuje modifikace pro různé účely. Ukázal se jako velmi výkonný a schopný překonat klasické sekvenční procedury včetně algoritmů plovoucího vyhledávání.

Na rozdíl od jiných metod je OS založeno na opakované modifikaci stávající podmnožiny příznaků X_d neměnné velikosti d. V tomto smyslu OS nezávisí na převládajícím směru vyhledávání. Toho je dosaženo střídáním tzv. *výkyvů dolů a nahoru*. Výkyvy oběma směry mají za cíl zlepšit stávající podmnožiny X_d nahrazením některých příznaků lepšími. *Výkyv dolů* nejprve odstraňuje a pak vrací zpátky, zatímco *výkyv nahoru* nejprve přidává a poté odstraňuje. Dva po sobě následující výkyvy tvoří *oscilační cyklus*. Na OS lze tedy nahlížet jako na řízenou posloupnost oscilačních cyklů. Hodnota o označovaná jako *hloubka oscilačního cyklu* určuje počet příznaků, které mohou být nahrazeny v jednom výkyvu. Hodnota o je algoritmem zvyšována po neúspěšných oscilačních cyklech a resetována na hodnotu 1 po každém zlepšení X_d.



Obr. 13.10 Grafy demonstrují vývoj velikosti zkoumaných podmnožin d-parametrizovaných algoritmů výběru příznaků: a) sekvenční dopředný výběr, b) sekvenční dopředný plovoucí výběr, c) oscilační vyhledávání.

Algoritmus je ukončen když o překročí uživatelem stanovený *limit* Δ. Průběh oscilačního vyhledávání je ilustrován ve srovnání s SFS a SFFS na obr. 13.10.

Každý algoritmus OS vyžaduje nějakou počáteční množinu d příznaků. Tuto počáteční množinu lze získat náhodně či jakýmkoliv jiným způsobem, tj. užitím některé z tradičních sekvenčních vyhledávacích procedur. Dále pro nahrazení o-tic příznaků v horních a dolních výkyvech lze použít téměř libovolnou proceduru výběru příznaků.

Oscilační vyhledávání (OS)

Algoritmus generuje podmnožinu d příznaků s rozsahem prohledávání omezeným parametrem $\Delta \geq 1$ (pro hledání v plném rozsahu položme $\Delta = D$).

KROK 1: *Inicializace.* Vycházíme z existující podmnožiny d příznaků X_d, kterou můžeme získat libovolným způsobem. Nastavme *hloubku oscilačního cyklu* $o = 1$.

KROK 2: Vygenerujme podmnožinu $X_d^{\downarrow} = ADD^o(REM^o(X_d))$.

KROK 3: Je-li $X_d^{\downarrow}$ lepší než X_d, položme $X_d = X_d^{\downarrow}$, $o = 1$, a jděme na Krok 2.

KROK 4: Vygenerujme podmnožinu $X_d^{\uparrow} = REM^o(ADD^o(X_d))$.

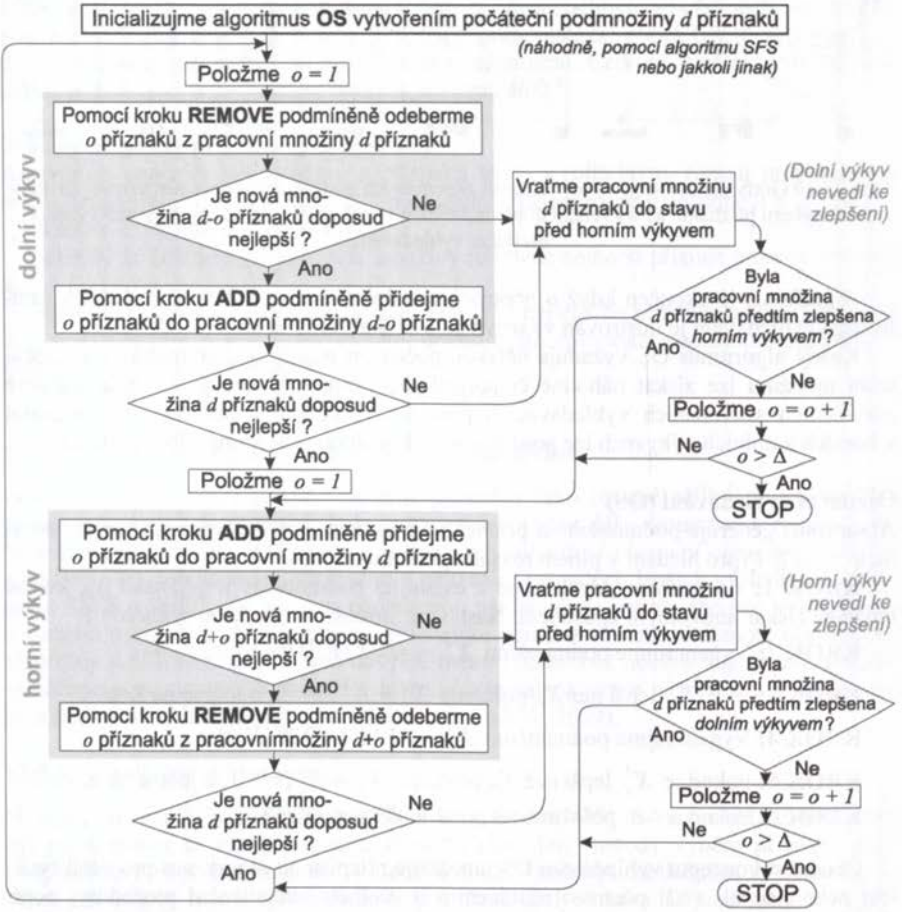
KROK 5: Pokud je $X_d^{\uparrow}$ lepší než X_d, položme $X_d = X_d^{\uparrow}$, $o = 1$, a jděme na Krok 2.

KROK 6: Pokud $o < \Delta$, položme $o = o + 1$ a jděme na Krok 2.

Obecnost konceptu vyhledávání OS umožňuje přizpůsobit algoritmus pro větší rychlost nebo naopak větší přesnost nastavením Δ, volbou inicializační procedury, popř. předefinováním operací *ADD* a *REM*. Na rozdíl od ostatních sekvenčních vyhledávacích procedur OS neztrácí čas vyhodnocováním podmnožin, jejichž kardinalita je příliš vzdá-

lena kardinalitě cílové (d). To zlepšuje schopnost OS nalézt dobrá řešení pro podmnožiny dané kardinality. Nejrychlejší zlepšování cílové podmnožiny lze očekávat v počátečních fázích algoritmu z důvodu malé počáteční hloubky oscilačních cyklů a tím i jejich rychlejšího průběhu. Později, když se stávající podmnožina příznaků přibližuje těsněji k optimu, cykly s malou hloubkou selhávají ve snaze zlepšit řešení a algoritmus proto rozšiřuje rozsah prohledávání ($o = o + 1$). I když se takto zvyšuje šance dostat se blíže k optimu, kompromis mezi šancí na nalezení lepšího řešení a růstem výpočetního času nabývá na významu. Toto chování algoritmu je nicméně velmi výhodné, protože dává možnost zastavit vyhledávání po určité omezené době, aniž by se tím riskovala ztráta výrazně lepšího výsledku. Hlavní výhody oscilačního vyhledávání lze shrnout takto:

– Lze na něj pohlížet jako na univerzální ladicí mechanismus schopný zlepšit řešení získaná jiným způsobem.



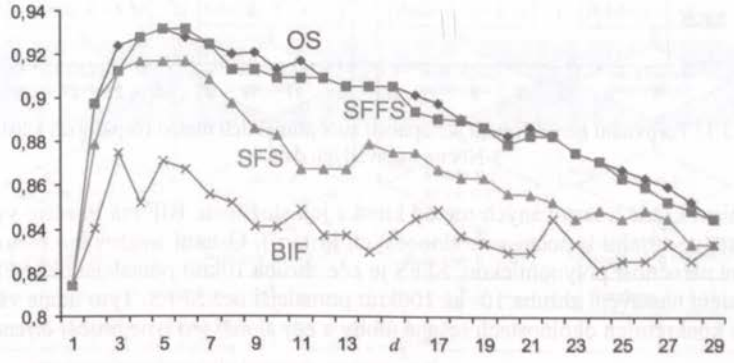
Obr. 13.11. Algoritmus oscilačního vyhledávání.

- Pro nízké hodnoty Δ a při použití rychlé inicializace (např. náhodné) je algoritmus OS velmi rychlý. V případě úloh o velké dimenzionalitě je jednou z mála alternativ k BIF. V práci (Novovičová a kol., 2006) je OS úspěšně použito při kategorizaci dokumentů k výběru řádově stovek příznaků při dimenzionalitě problému 10 000.
- Protože OS zpracovává od samého počátku podmnožiny o cílové kardinalitě, může nalézt řešení dokonce i v případech, v nichž *top-down* či *bottom-up* sekvenční procedury selžou kvůli numerickým problémům.
- Protože se řešení postupně zlepšuje po každém oscilačním cyklu s nejviditelnějším zlepšením v počátku, je možné algoritmus předčasně ukončit po předem specifikované době výpočtu a přitom dostat použitelné řešení. Z tohoto důvodu je OS použitelné v real-time systémech.
- Standardní sekvenční metody jsou náchylné k uvíznutí v lokálních extrémech. Opakovaný běh OS z několika různých náhodných počátečních bodů vyhledávacího prostoru dává větší šanci se takovému lokálnímu extrému vyhnout.

13.5.7 Experimentální porovnání *d*-parametrizovaných metod

Suboptimální *d*-parametrizované metody výběru příznaků, jež jsme popisovali postupně v odstavcích 13.5.1, 13.5.2, 13.5.4 a 13.5.6, byly uvedeny postupně od jednoduchých po komplexní. Metoda BIF (odst. 13.5.1) je nejrychlejší ale zároveň nejslabší metodou z hlediska maximalizace kriteriální funkce. OS poskytuje z uvedených metod nejsilnější optimalizační schopnost za cenu nejpomalejšího výpočtu (v závislosti na nastavení). Pro ilustraci uvedeného chování porovnáme výstup BIF, SFS, SFFS a OS na vzorové úloze výběru příznaků. Uvedené metody použijeme jakožto pouzdra (viz odst. 13.3.2).

Každou z uvedených metod testujeme na datech *ionosphere* (34 dim., 2 třídy: 225 a 126 vzorků) získaných z databáze UCI repository (Asuncion a kol., 2007), vybíráme nejlepší podmnožiny příznaků všech velikostí $d = 1,..., 34$. Data byla rozdělena na dvě části: 80 % jako trénovací a 20 % testovací. Metody byly použity k maximalizaci přesnosti klasifikace 3-NN klasifikátoru (*k*-nejbližších sousedů, viz (Dasarathy, 1991)), přesnost byla odhadována pomocí 10násobné krosvalidace v rámci trénovací části dat vždy na zkoumaném podprostoru.

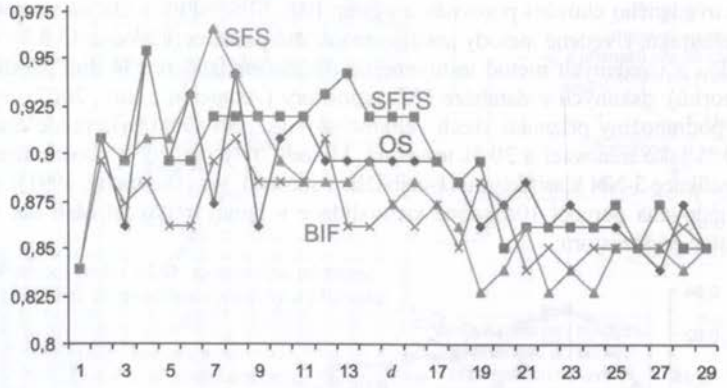


Obr. 13.12 Porovnání optimalizačního výkonu suboptimálních metod výběru příznaků (maximální dosažená úspěšnost klasifikace 3-NN na trénovacích datech).

BIF, SFS a SFFS nevyžadují žádné parametry, OS bylo nastaveno tak, aby každé vyhledávání proběhlo 15krát z různých náhodných počátečních podmnožin dané velikosti s $\Delta = 15$. Toto nastavení je sice velmi náročné na výpočetní čas ale dává šanci vyhnout se lokálním extrémům, ve kterých skončí výpočet většiny vyhledávacích algoritmů.

Obrázek 13.12 ukazuje maximální hodnotu kritéria dosaženou jednotlivými metodami pro každou velikost podmnožiny. Je vidět, že nejlepší výsledky dává ve většině případů OS, i když SFFS zůstává pozadu jen nepatrně. Optimalizační schopnost SFS se ukazuje jako podstatně menší, ale stále ještě podstatně větší než u BIF.

Pro praktické použití je však důležitější schopnost generalizace. Obrázek 13.13 ukazuje vliv nalezených řešení na úspěšnost klasifikace nezávislých testovacích dat. Z tohoto hlediska se rozdíly mezi uvažovanými metodami výrazně stírají. Komplexnější metody se ukazují jako náchylnější k přetrénování (viz oddíl 13.9), v případě OS je zřejmý největší rozdíl mezi úspěšností klasifikace na trénovacích a testovacích datech. Metoda SFFS se zde ukazuje jako v průměru nejspolehlivější, SFS poskytuje v tomto příkladu nejlepší izolovaný nezávislý výsledek. Za povšimnutí stojí, že ačkoliv největší optimalizované kriteriální hodnoty byly dosaženy pro podmnožiny o velikosti zhruba 6 příznaků, nejlepší výsledky na nezávislých datech lze pozorovat pro podmnožiny o různých velikostech mezi 4 až 13 příznaky. Tento příklad dobře ilustruje jeden z klíčových problémů výběru příznaků, že je obtížné najít podmnožiny, které dobře zobecňují. Poznamenejme, že uvedené výsledky jsou ilustrační. Při užití odlišné kriteriální funkce lze očekávat také odlišnosti v detailech funkčnosti jednotlivých metod.

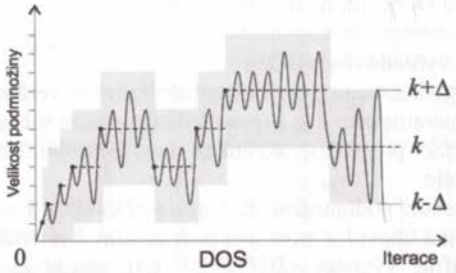


Obr. 13.13 Porovnání generalizační schopnosti suboptimálních metod (úspěšnosti klasifikace 3-NN na nezávislých datech).

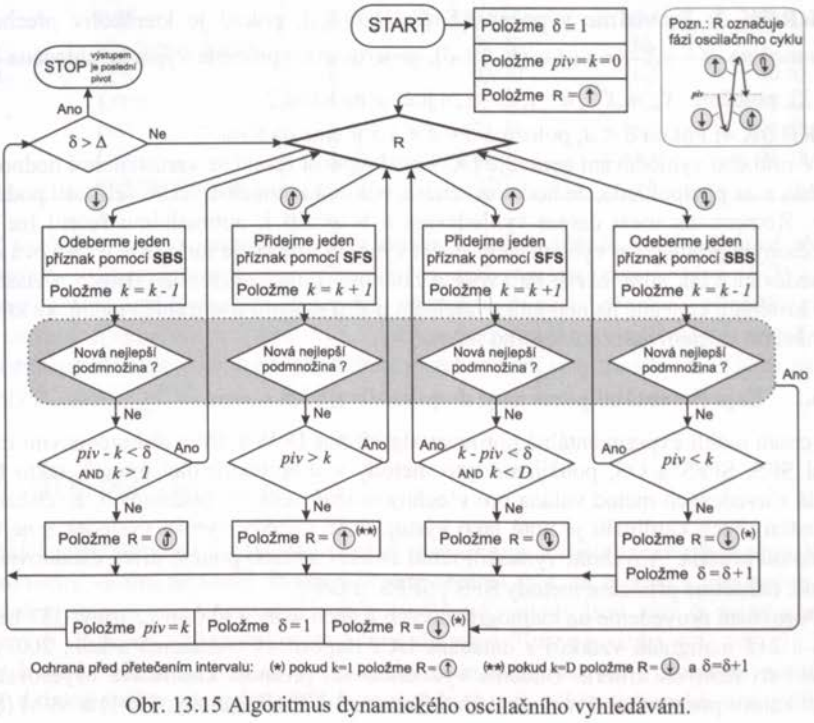
Rychlost každé z testovaných metod klesá s její složitostí. BIF má lineární výpočetní náročnost (ve vztahu k počtu vyhodnocených kritérií). Ostatní uvažované metody mají výpočetní náročnost polynomickou. SFFS je zde zhruba 10krát pomalejší než SFS. OS je při pomalém nastavení zhruba 10- až 100krát pomalejší než SFFS. Tyto údaje však silně závisí na konkrétních okolnostech řešené úlohy a zde slouží jen k nejhrubší orientaci.

13.6 Optimalizace velikosti podmnožiny – dynamické oscilační vyhledávání

Základní princip oscilačního vyhledávání (viz odst. 13.5.6) byl dále rozšířen do podoby *dynamického oscilačního vyhledávání* (Dynamic Oscillating Search, DOS) (Somol a kol., 2008b). Algoritmus DOS je iniciován libovolnou podmnožinou příznaků (včetně prázdné množiny). Podobně jako OS se opakovaně snaží zlepšit stávající množinu prováděním oscilačních cyklů (s rostoucí hloubkou oscilačního cyklu v případě neúspěchu).



Obr. 13.14 Graf demonstruje vývoj velikosti zkoumaných podmnožin *d*-optimalizujícího algoritmu výběru příznaků – dynamického oscilačního vyhledávání.



Obr. 13.15 Algoritmus dynamického oscilačního vyhledávání.

Velikost aktuálně vybrané podmnožiny se však v průběhu výpočtu mění a výpočet restartuje, kdykoliv je v libovolné fázi oscilačního cyklu nalezeno nové globálně nejlepší řešení. Na rozdíl od ostatních metod uváděných v této kapitole je tudíž DOS *d*-optimalizující procedurou.

Průběh dynamického oscilačního vyhledávání je ilustrován na obr. 13.14 (srovnejte s obr. 13.10). Podobně jako OS i DOS ukončuje výpočet, když stávající hloubka cyklu překročí uživatelem definovaný *limit* Δ. DOS také sdílí s OS stejné přednosti shrnuté v odst. 13.5.6, tedy schopnost doladit výsledky získané jiným způsobem, sekvenčnost zlepšování výsledku, nejrychlejší zlepšování v počátečních fázích vyhledávání atd. Blokové schéma algoritmu DOS vidíme na obr. 13.15.

Dynamické oscilační vyhledávání (DOS)
Algoritmus generuje podmnožinu příznaků optimalizované velikosti k s rozsahem prohledávání omezeným parametrem $\Delta \geq 1$ (pro hledání v plném rozsahu položme $\Delta = D$).

KROK 1: *Inicializaci* proveďme jedním ze dvou způsobů. Nastavme *hloubku oscilačního cyklu* $\delta = 1$. Dále

a) spočtěme výchozí podmnožinu $X_k = ADD(ADD\ (\emptyset))$, $k = 2$, nebo

b) můžeme vyjít z libovolné existující podmnožiny k příznaků X_k, pokud $k \geq 2$.

KROK 2: Proveďme výpočet $ADD^{\delta}(REM^{\delta}(X_k))$, pokud je kterákoliv přechodná podmnožina $X^{\downarrow}$ $|X^{\downarrow}| = i$, $i \in [k - \delta, k]$, generovaná v průběhu výpočtu, shledána lepší než X_k, položme $X_k = X^{\downarrow}$, $k = i$, $\delta = 1$, a restartujme Krok 2.

KROK 3: Proveďme výpočet $REM^{\delta}(ADD^{\delta}(X_k))$, pokud je kterákoliv přechodná podmnožina $X^{\uparrow}$ $|X^{\uparrow}| = j$, $j \in [k, k + \delta]$, generovaná v průběhu výpočtu, shledána lepší než X_k, položme $X_k = X^{\uparrow}$, $k = j$, $\delta = 1$, a jděme na Krok 2.

KROK 4: Pokud $\delta < \Delta$, položme $\delta = \delta + 1$ a jděme na Krok 2.

V průběhu vyhledávání generuje DOS posloupnost řešení se vzrůstajícími hodnotami kritéria a za předpokladu, že hodnota kritéria neklesá se zmenšující se velikostí podmnožiny. Kompromis mezi časem vyhledávání a blízkostí k optimálnímu řešení lze řídit předčasným přerušením vyhledávání. Počet výpočtů kritéria je řádu $O(n^3)$. Celková doba vyhledávání však silně závisí na zvolené hodnotě Δ, na konkrétních datech a vlastnostech kritéria a konečně na nepredikovatelném počtu restartů oscilačních cyklů, ke kterým dochází po každém nalezení lepšího řešení.

13.6.1 Experimentální porovnání *d*-optimalizujících metod

Abychom mohli experimentálně porovnat algoritmus DOS s dříve diskutovanými metodami SFS, SFFS a OS, použijeme tyto metody v *d*-optimalizující úpravě, takto bude každá z uvedených metod volána pro všechny možné velikosti podmnožin, ze získaných výsledků všech kardinalit je poté jako výstup dané metody vybrán výsledek s největší hodnotou kritéria. Abychom vyznačili tento odlišný způsob použití dříve diskutovaných metod, označíme příslušné metody SFS*, SFFS* a OS*.

Porovnání provedeme na mamogramových datech *wdbc* (30 dim., 2 třídy: 357 benigních a 212 maligních vzorků) z databáze UCI Repository (Asuncion a kol., 2007) při použití tří různých kritérií. Budeme vyhodnocovat přesnost klasifikace bayesovského klasifikátoru pro normální rozložení klasifikátoru 3-NN (Dasarathy, 1991) a SVM (Sup-

port Vector Machine) s jádrem radiální bázové funkce, viz (Chang a kol., 2001). Experimenty byly provedeny s použitím dvoustupňové krosvalidace. Vnější krosvalidační cyklus poskytuje různá dělení dat na trénovací a testovací část. Vnitřní krosvalidační cyklus pak dále rozděluje trénovací část dat na část pro trénink a validaci klasifikátoru. Výsledky experimentů jsou shrnuty v tabulce 13.3. Tabulka obsahuje tři oddíly udávající výsledky pro jednotlivé klasifikátory (kriteriální funkce). Sloupec I-CV ukazuje pro každou metodu výběru příznaků maximum dosažené hodnoty kritéria (průměr přes vnitřní krosvalidační cyklus). Sloupec O-CV vyjadřuje klasifikační přesnost na nezávislých testovacích datech (průměr přes vnější krosvalidační cyklus).

Tabulka 13.3 Porovnání *d*-optimalizujících suboptimálních metod (optimalizační výkonnost I-CV, generalizační schopnost O-CV, velikost nalezených podmnožin)

kritérium	metoda výběru	max. hodn. krit. (I-CV)	úspěšnost klasif. (O-CV)	velikost podmn.	doba výpočtu
Bayesovský klasifikátor	SFS*	0,962	0,933	10,8	00:00
	SFFS*	0,972	0,942	10,6	00:03
	OS*	0,970	0,940	9,9	00:06
	DOS	0,973	0,951	10,7	00:06
	všechny příznaky		0,945	30	
3-NN	SFS*	0,981	0,967	15,3	00:01
	SFFS*	0,983	0,970	13,7	00:09
	OS*	0,982	0,965	14,2	00:22
	DOS	0,984	0,965	12,4	00:31
	všechny příznaky		0,972	30	
SVM	SFS*	0,979	0,970	18,5	00:05
	SFFS*	0,982	0,968	16,2	00:23
	OS*	0,981	0,974	16,7	00:58
	DOS	0,983	0,968	12,8	01:38
	všechny příznaky		0,972	30	

Z výsledků lze vypozorovat, že dynamické oscilační vyhledávání je schopné překonat ostatní testované metody z hlediska schopnosti maximalizace kritéria (I-CV), má tendenci produkovat nejmenší podmnožiny příznaků, ale podobně jako v případě *d*-parametrizovaných experimentů (odst. 13.5.7) je zřejmé, že žádná z uvažovaných metod není jednoznačně nejlepší z hlediska generalizační schopnosti (úspěšnosti klasifikace na nezávislých datech O-CV).

13.7 Hybridní algoritmy

V předchozích oddílech jsme podali stručný přehled nejrůznějších nástrojů vhodných pro řešení úlohy výběru příznaků. Poukázali jsme také na fakt, že žádnou z uvedených metod nelze označit jako univerzálně nejlepší. Různé metody jsou více či méně vhodné v různých situacích. Příkladem může být rozdíl mezi filtry a pouzdry (viz odst. 13.3.2), filtry bývají podstatně rychlejší než pouzdra a umožňují tak řešit problémy podstatně větší dimenzionality, pouzdra však lépe umožňují při výběru příznaků vzít v úvahu

vlastnosti konkrétního klasifikátoru a mohou tudíž dosáhnout lepší klasifikační přesnosti. V takovéto situaci se nabízí idea zkombinovat vlastnosti obou uvažovaných přístupů.

Jednoduchý způsob definice *hybridní metody* výběru příznaků, kombinující výhody jiných existujících přístupů, byl popsán v práci (Liu a kol., 2005). Obecněji použitelný mechanismus definice hybridní metody, umožňující kombinovat libovolné dva způsoby vyhodnocování kriteriální funkce byl ukázán v článku (Somol a kol., 2006). V hybridních sekvenčních metodách definovaných tímto způsobem přepokládáme, že v každém kroku algoritmu je o přidání vhodného příznaku do pracovní množiny či odebrání z ní ven rozhodováno dvoufázově. Nejprve je využito rychlé *předfiltrovací kriteriální funkce* k redukci počtu kandidátů, teprve poté je mezi menším počtem kandidátů výsledný příznak vybrán pomocí pomalejší, ale jak předpokládáme vhodnější *hlavní kriteriální funkce*. Toto schéma lze implementovat v libovolných sekvenčních algoritmech výběru příznaků nahrazením definice 13.1 a definice 13.2 (kroky *ADD* a *REM* uvedené v odst. 13.5.2) upravenými definicemi, které uvádíme v dalším textu. Pro zjednodušení označme $J_F(.)$ rychleji vyhodnotitelné, ale pro daný problém méně vhodné *předfiltrovací kritérium*, dále označme $J_W(.)$ pomaleji vyhodnotitelné, ale pro danou úlohu předpokládané vhodnější *hlavní kritérium*. Dále zavedeme *hybridizační koeficient* $\lambda \in [0, 1]$ jakožto uživatelský parametr určující relativně podíl kandidátských příznaků zachovávaných po filtraci předběžným kritériem a následně vyhodnocovaných pomocí primárního kritéria. Poznamenejme, že v následujících definicích $\lceil \cdot \rceil$ označuje zaokrouhlení hodnoty.

Definice 13.3. Pro stávající množinu příznaků X_d a dané $\lambda \in [0, 1]$ nechť Z^+ je množina kandidátských příznaků

$$Z^+ = \{f_i : f_i \in Y \setminus X_d; i = 1,\ldots,\max\{1,\lceil \lambda \cdot |Y \setminus X_d| \rceil\}\} \tag{13.14}$$

taková, že

$$\forall f,g \in Y \setminus X_d, f \in Z^+, g \notin Z^+, \quad J_F^+(X_d,f) \geq J_F^+(X_d,g), \tag{13.15}$$

kde $J_F^+(X_d,f)$ označuje předfiltrovací kriterální funkci použitou pro vyhodnocení podmnožiny získané přidáním f do X_d, když $f \in Y \setminus X_d$. Nechť f^+ je takový příznak, že

$$f^+ = \arg\max_{f \in Z^+} J_W^+(X_d,f), \tag{13.16}$$

kde $J_W^+(X_d,f)$ označuje hlavní kriteriální funkci použitou pro vyhodnocení podmnožiny získané přidáním f do X_d, je-li $f \in Z^+$. Pak budeme říkat, že $ADD_H(X_d)$ je operace přidání příznaku f^+ do množiny X_d, abychom získali množinu X_{d+1}, jestliže

$$ADD_H(X_d) \equiv X_d \cup \{f^+\} = X_{d+1}, \quad X_d, X_{d+1} \subset Y. \tag{13.17}$$

Definice 13.4. Pro stávající množinu X_d a dané $\lambda \in [0, 1]$ nechť Z^- je množina kandidátských příznaků

$$Z^- = \{f_i : f_i \in X_d; i = 1,\ldots,\max\{1,\lceil \lambda \cdot |X_d| \rceil\}\} \tag{13.18}$$

taková, že

$$\forall f, g \in X_d, f \in Z^-, g \notin Z^-, \quad J_F^-(X_d, f) \geq J_F^-(X_d, g), \tag{13.19}$$

kde $J_F^-(X_d, f)$ označuje předfiltrovací kriteriální funkci použitou pro vyhodnocení podmnožiny získané vyloučením f z X_d, je-li $f \in X_d$. Nechť f^- je takový příznak, že

$$f^- = \arg\max_{f \in Z^-} J_W^-(X_d, f), \tag{13.20}$$

kde $J_W^-(X_d, f)$ označuje hlavní kriteriální funkci použitou pro vyhodnocení podmnožiny získané vyloučením f z X_d, když $f \in Z^-$. Pak budeme říkat, že $REM_H(X_d)$ je operace odstranění příznaku f^- z množiny X_d, abychom získali množinu X_{d-1}, jestliže

$$REM_H(X_d) \equiv X_d \setminus \{f^-\} = X_{d-1}, \quad X_d, X_{d-1} \subset Y. \tag{13.21}$$

Efekt hybridizace je ilustrován na příkladu z tabulky 13.4. Testovali jsme hybridizovanou metodu DOS (viz oddíl 13.6) na datech *waveform* (40 dim., 2 třídy: 1692 a 1653 vzorků) z databáze UCI Repository (Asuncion a kol., 2007). Jakožto předfiltrovací kritérium bylo použito Bhattacharyovy vzdálenosti (Devijver a kol., 1982). Hlavním kritériem byl odhad úspěšnosti klasifikace 3-NN klasifikátoru (*k*-Nearest Neigbor, neboli *k*-nejbližších sousedů, viz (Dasarathy, 1991)).

Uvedené výsledky ukazují nárůst získané hodnoty kritéria, ale také výrazný nárůst doby výpočtu pro rostoucí hodnoty λ, jenž odpovídá rozšiřování počtu kandidátských příznaků vyhodnocovaných pomocí hlavního kritéria. Ověřením na nezávislých datech se však v tomto případě ukázalo, že hybridizací lze nejen ušetřit výpočetní čas, ale i zlepšit chování klasifikátoru na neznámých datech. Nejlepší výsledek zde odpovídá nejnižší hodnotě λ. Je zřejmé že v daném případě vede maximalizace hlavního kritéria metodou DOS k přeučení a že předfiltrovací kritérium zde přeučení úspěšně omezuje.

Tabulka 13.4 Porovnání úspěšnosti hybridizovaného algoritmu dynamického oscilačního vyhledávání pro různé koeficienty hybridizace

koeficient hybridizace λ	0,01	0,25	0,5	0,75	1
maximální hodnota kritéria	0,907	0,913	0,921	0,921	0,921
přesnost klasifikace na nezáv. datech	0,916	0,911	0,911	0,910	0,910
nalezená velikost podmnožiny	11	10	15	17	17
výpočetní čas	1:12	8:06	20:42	35:21	48:24

13.8 Výběr příznaků založený na modelu směsi hustot

Alternativou k vyhledávacím metodám výběru příznaků typu filtr či pouzdro (viz odst. 13.3.2) jsou tzv. metody integrované (embedded), v nichž je proces výběru příznaků přímo součástí učícího procesu, resp. odhadu modelu.

Uvažujme nyní problém výběru příznaků v situacích, kdy vícerozměrné podmíněné hustoty $p(\boldsymbol{x} \mid \omega_j)$ a často i apriorní pravděpodobnosti $P(\omega_j)$ jsou buď jen částečně známé, nebo neznámé a jediným zdrojem informace se stávají soubory nezávislých pozorování z jednotlivých tříd, tj. máme k dispozici trénovací množiny T_ω se vzorky ze třídy ω.

Odhadování apriorních pravděpodobností většinou není problém, obvykle je nahrazujeme relativními četnostmi výskytu vektorů příznaků z jednotlivých tříd v trénovací množině, nebo známe odhady jejich výskytu v praxi. Některé nebo všechny podmíněné hustoty je vhodné modelovat jako konečné směsi hustot pravděpodobnosti, které jsou flexibilnější než jednoduché pravděpodobnostní modely. Pomocí modelu směsí lze reprezentovat libovolně složité hustoty, viz např. (Hastie a kol., 1996), (Palm, 1994), (McLachlan, 2004), (Ripley, 2005) a (Webb, 2002).

V publikacích (Pudil a kol., 1994b), (Pudil a kol., 1995), (Novovičová a kol., 1996) a (Novovičová a kol., 1998) je uveden přístup k výběru příznaků založený na aproximaci neznámých podmíněných hustot pravděpodobnosti modifikovanou konečnou směsí součinových komponent.

13.8.1 Konečná směs hustot pravděpodobnosti

Konečná směs hustot pravděpodobnosti pro podmíněnou hustotu ve třídě ω je definována ve tvaru

$$p(\boldsymbol{x} \mid \alpha_\omega, \theta_\omega) = \sum_{m=1}^{M_\omega} \alpha_m^\omega p(\boldsymbol{x} \mid \theta_m^\omega), \quad \sum_{m=1}^{M_\omega} \alpha_m^\omega = 1, \quad \alpha_m^\omega > 0, \tag{13.22}$$

kde M_ω označuje počet komponent směsi a $p(\boldsymbol{x} \mid \theta_m^\omega)$ je hustota m-té komponenty ve třídě ω, α_m^ω udává váhu m-té komponenty směsi, θ_m^ω značí parametr m-té komponenty směsi a $\{\alpha_\omega, \theta_\omega\} = \{\alpha_1^\omega, ..., \alpha_{M_\omega}^\omega, \theta_1^\omega, ..., \theta_{M_\omega}^\omega\}$ vyjadřuje množinu všech parametrů směsi. Tento model předpokládá, že každá komponenta směsi má vícerozměrné rozdělení s vlastními parametry. Obvykle uvažujeme, že komponenty mají stejný parametrický tvar (např. vícerozměrný normální). Model (13.22) může být zjednodušen za předpokladu, že pro každou komponentu $p(\boldsymbol{x} \mid \theta_m^\omega)$ směsi (13.22) jsou příznaky statisticky nezávislé. Pak je každá komponenta součinem jednorozměrných marginálních hustot $p(x_i \mid \theta_m^\omega)$ a směs (13.22) lze přepsat do jednoduššího tvaru

$$p(\boldsymbol{x} \mid \alpha_\omega, \theta_\omega) = \sum_{m=1}^{M_\omega} \alpha_m^\omega \prod_{i=1}^{D} p(x_i \mid \theta_{mi}^\omega), \tag{13.23}$$

v němž θ_{mi}^ω je parametr m-té komponenty ve třídě ω odpovídající příznaku x_i. To znamená že $p(x_i \mid \theta_m^\omega)$ je hustota i-tého příznaku v m-té komponentě hustoty ve třídě ω a podmíněná hustota pravděpodobnosti je modelována jako směs nezávislých pravděpodobnostních modelů.

13.8.2 Modifikovaná konečná směs součinových komponent

V popisovaném přístupu ke globálnímu výběru příznaků předpokládáme, že některé příznaky nejsou významné pro strukturu směsi (13.23) v tom smyslu, že příznak x_i je *nevýznamný* pro strukturu směsi (13.23), jestliže jeho rozdělení pravděpodobnosti je určeno *společnou* (*sdílenou*) *hustotou* napříč všemi komponentami, a tudíž je nezávislé na m, tj. $p(x_i \mid \theta_m^\omega) = g_0(x_i \mid \theta_{0i})$. Příznak x_i je *významný* pro strukturu směsi (13.23),

jestliže jeho rozdělení pravděpodobnosti je určeno *specifickou hustotou*, tj. $p(x_i \mid \theta_m^\omega) = g(x_i \mid \theta_{mi}^\omega)$. Nechť $\Phi = (\phi_1, \phi_2, ..., \phi_D) \in \{0, 1\}^D$ je množina binárních parametrů a dále

$\phi_i = 1$ znamená, že příznak x_i je *významný* a měl by být modelován použitím *specifické komponenty* směsi,

$\phi_i = 0$ znamená, že příznak x_i je *nevýznamný* a měl by být reprezentován *společnou (sdílenou) hustotou*.

Směs hustot v (13.23) lze nyní přepsat do tvaru

$$p(\boldsymbol{x} \mid \alpha_\omega, \theta_\omega, \theta_0, \Phi) = \sum_{m=1}^{M_\omega} \alpha_m^\omega p(\boldsymbol{x} \mid \theta_\omega, \theta_0, \Phi) = \sum_{m=1}^{M_\omega} \alpha_m^\omega \prod_{i=1}^{D} [g_0(x_i \mid \theta_{0i})^{1-\phi_i} g(x_i \mid \theta_{mi}^\omega)^{\phi_i}], \quad (13.24)$$

kde $\theta_0 = (\theta_{01}, ..., \theta_{0D})$. Takto lze pohlížet na binární parametry ϕ_i jako na *řídicí veličiny*, neboť složitost a struktura směsi (13.23) může být řízena pomocí těchto parametrů.

Výpočet odhadů parametrů $(\alpha, \theta, \theta_0, \Phi) = \{(\alpha_\omega, \theta_\omega, \theta_0, \Phi, \omega \in \Omega\}$ směsi (13.24) maximalizujících pro dané trénovací množiny T_ω, $\omega \in \Omega$, globální logaritmickou věrohodnostní funkci

$$L(\alpha, \theta, \theta_0, \Phi) = \sum_{\omega \in \Omega} \frac{P(\omega)}{|\mathcal{T}_\omega|} \sum_{\mathbf{x} \in \mathcal{T}_\omega} \sum_{m=1}^{M_\omega} \log p(\boldsymbol{x} \mid \alpha_\omega, \theta_\omega, \theta_0, \Phi) \quad (13.25)$$

je založen na algoritmu EM (Expectation-Maximization) (Redner a kol., 1984).

• *E-krok*: Pro dané parametry $\alpha_m^\omega, \theta_m^\omega, \theta_0, \Phi$ a pro $m = 1, ..., M_\omega$, $\boldsymbol{x} \in T_\omega$, vypočteme aposteriorní pravděpodobnosti

$$p(m \mid \boldsymbol{x}, \omega) = \frac{\alpha_m^\omega p(\boldsymbol{x} \mid \theta_m^\omega, \theta_0, \Phi)}{\sum_{j=1}^{M_\omega} \alpha_j^\omega p(\boldsymbol{x} \mid \theta_j^\omega, \theta_0, \Phi)} = \frac{\alpha_m^\omega \prod_{i=1}^{D} [g_0(x_i \mid \theta_{0i})^{1-\phi_i} g(x_i \mid \theta_{mi}^\omega)^{\phi_i}]}{\sum_{j=1}^{M_\omega} \alpha_j^\omega \prod_{i=1}^{D} [g_0(x_i \mid \theta_{0i})^{1-\phi_i} g(x_i \mid \theta_{ji}^\omega)^{\phi_i}]}. \quad (13.26)$$

Pro tyto pravděpodobnosti určíme očekávanou globální logaritmickou věrohodnostní funkci

$$L(\alpha, \theta, \theta_0, \Phi) = \sum_{\omega \in \Omega} \frac{P(\omega)}{|\mathcal{T}_\omega|} \sum_{\mathbf{x} \in \mathcal{T}_\omega} \sum_{m=1}^{M_\omega} p(m \mid \boldsymbol{x}, \omega) \log \alpha_m^\omega p(\boldsymbol{x} \mid \theta_\omega, \theta_0, \Phi). \quad (13.27)$$

• *M-krok*: Při pevných aposteriorních pravděpodobnostech (13.26) vypočteme nové hodnoty parametrů směsi

$$(\hat{\alpha}, \hat{\theta}, \hat{\theta}_0, \hat{\Phi}) = \arg \max_{\alpha, \theta, \theta_0, \Phi} \{L(\alpha, \theta, \theta_0, \Phi)\}. \quad (13.28)$$

Odhad binárních parametrů ϕ_i, $i = 1, ..., D$, je zahrnut do M-kroku algoritmu EM a závisí na zvoleném kritériu významnosti příznaků.

Poznamenejme, že v algoritmu EM předpokládáme na počátku iniciované hodnoty parametrů. Obvyklým způsobem inicializace je nějaká forma náhodné inicializace, podrobnější údaje najdeme v práci (McKenzie a kol., 1994).

Výpočet se značně zjednoduší za předpokladu že:
– jednorozměrné hustoty $g(x_i \mid \theta_{mi}^{\omega})$ a $g_0(x_i \mid \theta_{0i})$ patří do rodiny jednorozměrných normálních hustot,
– $g_0(x_i \mid \theta_{0i})$ má stejné parametry pro všechny třídy $\omega \in \Omega$,
– počet komponent směsi (13.24) a počet příznaků je předem stanoven.

Protože zde není možné prezentovat úplný podrobný popis zmíněných metod, odkazujeme čtenáře na původní zdroje (Pudil a kol., 1995), (Novovičová a kol., 1996). Základní myšlenka modelu (13.24) inspirovala řadu odborníků k dalšímu vývoji tohoto postupu, např. (Miller a kol., 2003), (Law a kol., 2004), (Krishnapuram a kol., 2004), (Graham a kol., 2006), (Bouguila a kol., 2009) a (Bouguila, 2010).

13.8.3 Míry významnosti příznaků při užití směsového modelu

Navržený model směsi (13.24) umožňuje odvodit dvě metody výběru příznaků, které se liší volbou kritéria vyhodnocování významnosti příznaků.

Aproximační metoda pomocí odhadnutých řídicích veličin, tj. vektoru $\hat{\Phi}_d$ s d nenulovými složkami, vyhledá příznaky, které dávají množinu aproximací podmíněných hustot pravděpodobnosti nejlepší ve smyslu minimalizace směsi Kullbackových–Leiblerových vzdáleností mezi skutečnou a předpokládanou podmíněnou hustotou vektoru $\boldsymbol{x}$ ve třídě ω s váhami $P(\omega_1),\ldots, P(\omega_C)$. Příznaky jsou vybírány s ohledem na reprezentační schopnost, která v některých případech nemusí znamenat diskriminační schopnost.

Divergenční metoda určí ty příznaky, které jsou nejlepší ve smyslu maximalizace Kullbackovy J-divergence (Boekee a kol., 1979) mezi dvěma třídami. Cílem metody je vybrat ty příznaky, které jsou nejlepší ve smyslu rozlišení mezi třídami.

13.8.4 Odvození rozhodovacího pravidla a souhrn vlastností směsového přístupu

Popsali jsme řešení problému výběru příznaků, při kterém místo výběru vhodné podmnožiny příznaků pomocí vyhledávacích metod odhadujeme množinu binárních veličin (jednu pro každý příznak), jež charakterizují významnost každého příznaku. Tento odhad je proveden pomocí algoritmu EM.

Uvažujme problém klasifikace daného objektu popsaného vektorem $\boldsymbol{x}$ do jedné třídy z množiny Ω. To je možné pomocí Bayesova rozhodovacího pravidla s minimální chybou:

Zařaď $\boldsymbol{x}$ do třídy ω_l, jestliže

$$\omega_l = \arg\max_{j=1,\ldots,C} \frac{P(\omega_j)p(\boldsymbol{x} \mid \omega_j)}{p(\boldsymbol{x})},\ l \in \{1,\ldots,C\},\quad p(\boldsymbol{x}) = \sum_{j=1}^{C} P(\omega_j)p(\boldsymbol{x} \mid \omega_j), \tag{13.29}$$

kde $p(\boldsymbol{x})$ je nepodmíněná hustota pravděpodobnosti rozložení vektoru $\boldsymbol{x}$. Dosadíme-li v uvedeném smyslu nalezenou optimální aproximaci podmíněné hustoty do Bayesova pravidla (13.29), společná hustota g_0 může být eliminována a nový objekt reprezentovaný vektorem $\boldsymbol{x}$ lze klasifikovat bezprostředně do jedné z možných tříd pouze na základě d příznaků $x_{i_1},\ldots,x_{i_d}$, kde $\{i_1,\ldots, i_d\}$ je permutace čísel $\{1,\ldots, D\}$, a to určením odhadu vektoru $\hat{\Phi}_d$ o d nenulových složkách. Tento klasifikátor v redukovaném příznakovém prostoru nazýváme *pseudoBayesův* klasifikátor.

Popsaný přístup k výběru příznaků je vhodný zvláště pro případy, když máme k dispozici dostatečně velkou trénovací množinu a zároveň některé nebo všechny podmíněné hustoty pravděpodobnosti jsou vícemodální, nebo když se předpokládá existence podtříd v každé třídě. Popsaný přístup může sloužit k více cílům:
- určení struktury vícerozměrných rozdělení,
- určení nejdůležitějších příznaků a tím možnosti snížit dimenzionalitu,
- odvození rozhodovacího pravidla založeného na vybraných příznacích pro řešení problému klasifikace do dvou nebo více tříd.

13.8.5 Experiment na reálných datech

Za pomoci metod výběru příznaků založených na modelu směsi (13.24) jsme provedli experimenty na datech *speech* a *wdbc* použitých již v odstavcích 13.4.6.2 a 13.6.1. V tomto experimentu bylo 67 % dat použito pro trénink a 33 % pro test úspěšnosti klasifikace. Experimenty jsme uskutečnili při dvou různých způsobech inicializace: náhodné inicializaci a inicializaci typu psi a králíci (McKenzie a kol., 1994). Tabulka 13.5 demonstruje potenciál směsové aproximační metody – s 5 komponentami (sloupec aprox. 5 k.) pro data *speech* a s 1, 5 či 20 komponentami pro data *wdbc*, metoda dosahuje lepší úspěšnosti klasifikace než gaussovský klasifikátor, který je založen na předpokladu unimodality dat.

Tabulka 13.5 Porovnání úspěšnosti klasifikace gaussovského a pseudobayesovského klasifikátoru (pro aproximační metodu) za použití různých počtů komponent směsi

		gauss. klasif.	aprox. 1 k.	aprox. 5 k.	aprox. 10 k.	aprox. 20 k.
speech	(náhodná inicializace)	0,916	0,784	0,924	0,908	0,910
	(inicializace psi a králíci)	–	0,784	0,926	0,936	0,916
wdbc	(náhodná inicializace)	0,940	0,947	0,947	0,940	0,954
	(inicializace psi a králíci)	–	0,947	0,947	0,940	0,940

13.9 Problém přeučení a problém stability výběru příznaků

Ve starší literatuře bylo zvykem ohodnocovat účinnost metod výběru příznaků na základě schopnosti najít optimum nebo hodnotu co nejbližší k optimu vzhledem ke zvolené kriterální funkci. V poslední době je důraz kladen na vyhodnocení schopnosti generalizovat, tj. schopnosti nalézt takové příznaky, pro něž výsledné rozhodovací pravidlo dává nejlepší výsledky na nezávislých datech. Bylo ukázáno, že podobně jako při konstrukci klasifikátorů je v případě výběru příznaků nutné zabránit přetrénování, tj. přílišnému přizpůsobení výsledku detailním vlastnostem trénovacích dat, které však mohou být zavádějící, zejména nemají-li trénovací data dostatečnou velikost (Raudys, 2006). Problém přetrénování je ilustrován poklesem úspěšnosti klasifikace na obr. 13.12 v porovnání s obrázkem 13.13. Testování účinnosti výběru příznaků na nezávislých testovacích datech je důležité také při porovnávání jednotlivých metod (Reunanen, 2003).

Problém porovnávání metod výběru příznaků mezi sebou je v literatuře chápán dvojznačně. Je rozdíl, porovnáváme-li samotnou optimalizační schopnost vyhledávacích procedur nebo účinnost klasifikátoru při použití metod výběru ve specifickém kontextu.

Konečná účinnost klasifikátoru je nejdůležitější mírou kvality, je však kontextově závislá a obvykle není podle ní vhodné přijímat zobecňující závěry o jednoznačné nadřazenosti či podřazenosti jedněch metod oproti druhým.

V literatuře panuje obecná shoda, že výběr příznaků založený na pouzdrech vede k přesnějším klasifikátorům než výběr příznaků založený na filtrech (viz odst. 13.3.2). Zároveň je však třeba zdůraznit, že pouzdra bývají náchylnější k přetrénování. Naopak slabší vztah mezi kriteriálními funkcemi založenými na filtrech a přesnosti konkrétního klasifikátoru může pomoci k lepšímu zobecnění a tím k lepší funkčnosti klasifikátoru na neznámých datech. Podobně lze očekávat větší robustnost výsledků filtrového výběru příznaků při použití v kontextu více různých klasifikátorů.

Problém odhadu účinnosti klasifikátoru není v žádném případě jednoduchý. Existuje mnoho strategií odhadování, jejichž vhodnost je závislá na problému (resubstituce, dělení dat, metody hold-out, krosvalidace, metoda leave-one-out atd.). Podrobnou studií o problémech souvisejících s učením klasifikátorů stabilizací slabých klasifikátorů a vyhodnocováním jejich přesnosti je (Skurichina, 2001).

13.9.1 Problém stability výběru příznaků

Výběr příznaků může být zatížen také problémem stability. Zdánlivě uspokojivý výsledek výběru příznaků tak může být ve skutečnosti výsledkem náhody, nestabilní metoda pak může při nepatrné změně okolností vést k výsledku zcela odlišnému. Je zřejmé, že nestabilní účinnost procesu FS by mohla vážně zhoršit vlastnosti konečného klasifikátoru výběrem špatných příznaků.

V souladu s prací (Kalousis a kol., 2007) definujeme *stabilitu algoritmu* výběru příznaků jakožto robustnost preferencí příznaků proti vlivu různých trénovacích množin generovaných ze stejného rozdělení. Stabilitu algoritmů FS budeme zkoumat v podobě jejich preference příznaků přes vybrané podmnožiny $S \subseteq Y$.

Současné práce zabývající se stabilitou metod FS se hlavně zaměřují na různé indexy stability zavádějící míry založené na Hammingově vzdálenosti (Dunne a kol., 2002), korelačních koeficientech a Tanimotově vzdálenosti (Kalousis a kol., 2007), indexu konzistence (Kuncheva, 2007) a Shannonově entropii (Křížek a kol., 2007). Poznamenejme, že stabilita procedury FS závisí na velikosti trénovací množiny, kritériu použitém k ohodnocení výběru příznaků a složitosti procedury FS (Raudys, 2006). V dalším výkladu se zaměříme na některé nové míry umožňující posoudit nejen stabilitu *d*-parametrizovaných ale i *d*-optimalizujících metod FS (Somol a kol., 2008a).

13.9.2 Vybrané míry stability výběru příznaků

Nechť $\mathbf{S} = \{S_1,..., S_n\}$ je systém n podmnožin příznaků

$$S_j = \{f_{k_i} \mid i = 1,...,d_j, f_{k_i} \in Y, d_j \in \{1,...,|Y|\}\}, j = 1,...,n,\ n > 1, n \in N,$$

získaných z n opakovaných volání zkoumaného algoritmu výběru příznaků na různých trénovacích množinách vybraných z daného souboru dat. Naším cílem je definovat míru stability takovou, aby ohodnocovala systém $\mathbf{S}$ hodnotou z intervalu [0, 1], kde větší hodnota označuje větší stabilitu. Hodnota 0 nechť označuje situaci, kdy všechny podmnožiny příznaků generované zkoumanou metodou jsou disjunktní, a hodnota 1 nechť označuje situaci, kdy všechny podmnožiny příznaků jsou identické.

Nechť X je podmnožina množiny Y reprezentující všechny příznaky, které se objeví v kterékoliv množině v systému S:

$$X = \{f \mid f \in Y, F_f > 0\} = \bigcup_{i=1}^{n} S_i, \quad X \neq \emptyset, \tag{13.30}$$

kde F_f je počet výskytů (frekvence) příznaku $f \in Y$ v systému S. Nechť N značí celkový počet výskytů každého příznaku v systému S, tj.

$$N = \sum_{g \in X} F_g = \sum_{i=1}^{n} |S_i|, \quad N \in \mathbf{N}, \quad N \geq n. \tag{13.31}$$

Definice 13.5. Vážená konzistence $CW(\mathsf{S})$ systému S je definována jako

$$CW(\mathsf{S}) = \sum_{f \in X} w_f \frac{F_f - F_{\min}}{F_{\max} - F_{\min}}, \tag{13.32}$$

kde $w_f = \dfrac{F_f}{N}$, $0 < w_f \leq 1, \sum_{f \in X} w_f = 1.$

Vzhledem k tomu, že $F_f = 0$ pro všechna $f \in Y \setminus X$, *vážená konzistence* $CW(\mathsf{S})$ může být ekvivalentně vyjádřena s použitím označení (13.31) ve tvaru

$$CW(\mathsf{S}) = \sum_{f \in X} \frac{F_f}{N} \cdot \frac{F_f - F_{\min}}{F_{\max} - F_{\min}} = \sum_{f \in Y} \frac{F_f}{N} \cdot \frac{F_f - 1}{n - 1}. \tag{13.33}$$

Je zřejmé, že $CW(\mathsf{S}) = 0$ tehdy a jen tehdy, když $N = |X|$, tj. tehdy a jen tehdy, když $F_f = 1$ pro všechna $f \in X$. Taková situace není ale myslitelná ve většině reálných úloh. Kdykoliv $n > |X|$, některé příznaky se musí objevit ve více než jedné podmnožině, a tudíž $CW(\mathsf{S}) > 0$. Podobně $CW(\mathsf{S}) = 1$ tehdy a jen tehdy, když $N = n|X|$, jinak všechny podmnožiny nemohou být identické.

Je zřejmé, že pro každé N, n reprezentující nějaký systém podmnožin S a pro dané Y existuje systém $\mathsf{S}_{\min}$ s takovou konfigurací příznaků ve svých podmnožinách, že dává minimální možnou hodnotu, označenou $CW_{\min}(N, n, Y)$, která může být větší než 0. Podobně existuje systém $\mathsf{S}_{\max}$, který dává maximální možnou hodnotu $CW(.)$, označíme ji $CW_{\max}(N, n)$, jež může být menší než 1.

Je zřejmé, že $CW_{\min}(.)$ je větší, pokud se velikosti podmnožin příznaků v systému blíží celkovému počtu příznaků $|Y|$, neboť v takovém systému jsou si podmnožiny navzájem podobnější. Proto použití míry (13.32) pro porovnání stability různých metod FS může vést k zavádějícím závěrům, mají-li porovnávané metody tendenci generovat systémy podmnožin výrazně odlišné průměrné velikosti. Tento problém budeme označovat jako problém zkreslení velikosti podmnožiny. Poznamenejme, že většina měr stability, které jsou k dispozici, je tímto problémem ovlivněna. Z tohoto důvodu zavádíme další míru nazvanou *relativní vážená konzistence*, která potlačuje vliv velikostí podmnožin v systému na výslednou hodnotu míry stability.

Definice 13.6. Relativní vážená konzistence $CW_{rel}(\mathsf{S}, Y)$ systému S charakterizovaného veličinami N, n a daným Y je definovaná jako

$$CW_{rel}(\mathsf{S},Y) = \frac{CW(\mathsf{S}) - CW_{min}(N,n,Y)}{CW_{max}(N,n) - CW_{min}(N,n,Y)}, \tag{13.34}$$

kde $CW_{rel}(\mathsf{S}, Y) = CW(\mathsf{S})$, pokud $CW_{max}(N, n) = CW_{min}(N, n, Y)$.

Označme pro jednoduchost $G = (N \bmod |Y|)$ a $H = (Nm \bmod n)$. V práci (Somol a kol., 2008a) bylo ukázáno, že

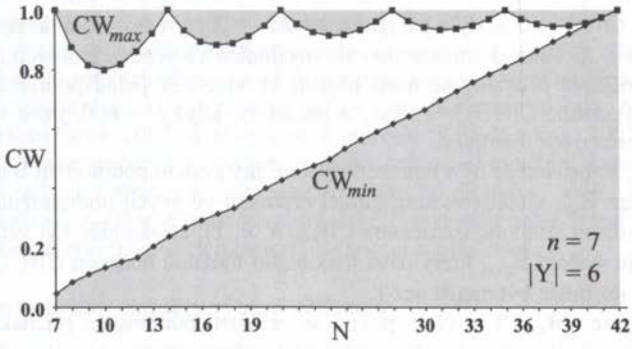
$$CW_{min}(N,n,Y) = \frac{N^2 - |Y|(N-G) - G^2}{|Y|N(n-1)} \tag{13.35}$$

a

$$CW_{max}(N,n) = \frac{H^2 + N(n-1) - Hn}{N(n-1)}. \tag{13.36}$$

Relativní vážená konzistence pak nabývá tvar

$$CW_{rel}(\mathsf{S},Y) = \frac{|Y|\left(N - G + \sum_{f \in Y} F_f(F_f - 1)\right) - N^2 + G^2}{|Y|\left(H^2 + n(N-H) - G\right) - N^2 + G^2}. \tag{13.37}$$



Obr. 13.16 Ilustrace mezí míry CW.

Meze vážené konzistence $CW_{max}(N, n)$ a $CW_{min}(N, n, Y)$ jsou ilustrovány na obr. 13.16. Poznamenejme, že CW_{rel} může být citlivá na malé změny systému, když se N blíží k maximu (pro dané N, n a n).

Lze ukázat, že pro každé N, n reprezentující nějaký systém podmnožin S a pro dané Y platí, že $0 \le CW_{rel}(\mathsf{S},Y) \le 1$ a pro odpovídající systémy S_{min} a S_{max} platí, že $CW_{rel}(\mathsf{S}_{min}) = 0$ a $CW_{rel}(\mathsf{S}_{max}) = 1$.

Míra (13.34) nevykazuje nežádoucí chování dávající větší hodnoty pro systémy s velikostí podmnožin blízkou $|Y|$, tj. je nezávislá na velikosti podmnožin příznaků vybíraných zkoumanými metodami FS při pevném Y. Můžeme říci, že tato míra charakterizuje pro dané S, Y relativní náhodnost složení množin v systému S na stupnici mezi maximální a minimální hodnotou vážené konsistence (13 32).

Koncepčně odlišná míra je definována v práci (Kalousis a kol., 2007). Je odvozena z *Tanimotova indexu* (*koeficientu*) definovaného jako podíl velikosti průniku a velikosti sjednocení podmnožin S_i a S_j (Duda a kol., 2000)

$$S_K(S_i, S_j) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|}. \tag{13.38}$$

Definice 13.7. Průměrný Tanimotův index systému S je definován výrazem

$$ATI(\mathsf{S}) = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} S_K(S_i, S_j). \tag{13.39}$$

ATI(S) je průměrná míra podobnosti přes všechny páry podmnožin příznaků v S. Nabývá hodnot z intervalu [0, 1], přičemž 0 indikuje prázdný průnik mezi všemi páry podmnožin S_i, S_j, a 1 indikuje, že všechny podmnožiny systému S jsou identické.

13.9.3 Experimenty s mírami stability

Pro ilustraci uváděných měr stability jsme provedli několik experimentů. Budeme porovnávat *d*-optimalizující metodu DOS a metody BIF, SFS a SFFS v *d*-optimalizující úpravě (úprava viz odst. 13.6.1). Pro odlišení od standardního *d*-parametrizovaného průběhu vyhledávání označíme tyto metody BIF*, SFS* a SFFS*. Podobně jako v odst. 13.6.1 porovnání provedeme na mamogramových datech *wdbc* (30 dim., 2 třídy: 357 benigních a 212 maligních vzorcích) z databáze UCI Repository (Asuncion a kol., 2007) při použití tří různých kritérií. Budeme vyhodnocovat přesnost klasifikace bayesovského klasifikátoru pro normální rozložení, klasifikátoru 3-NN (Dasarathy, 1991) a SVM (Support Vector Machine) s jádrem radiální bázové funkce, viz (Chang a kol., 2001). Pro každou kombinaci metoda–klasifikátor byl výběr příznaků opakován 1000krát na vždy jinak náhodně vybrané datové podmnožině odpovídající 80 % původních dat. Při každém opakování výběru příznaků byly 2/3 této datové podmnožiny použity k výběru příznaků (kritérium v podobě odhadu úspěšnosti klasifikace vyhodnocováno pomocí 10násobné krosvalidace) a zbývající 1/3 použita pro testování.

Výsledky shrnujeme v tabulce 13.6. Všechny míry, *CW*, CW_{rel} a *ATI*, indikují metodu BIF* jakožto obecně nejstabilnější metodu výběru příznaků, což potvrzuje závěry z článku (Kuncheva, 2007). Porovnáme-li mezi sebou ostatní zkoumané metody, nejlepší stabilitu i nejmenší velikost nalézaných podmnožin v daném experimentu vykazuje DOS. Podobným způsobem je možné zkoumat vhodnost konkrétních metod pro konkrétní úlohy. Poznamenejme, že CW_{rel} je zde jedinou mírou, která správně odhaluje náhodný výběr příznaků (v tabulce hodnoty blízké 0).

Tabulka 13.6 Experimentální vyhodnocení stability metod výběru příznaků typu pouzdro

		Přesnost klasif.		Velikost podmn.		CW	CW_{rel}	ATI	Doba výpočtu
Kritérium	Metoda	Průměr	*St.odch*	Průměr	*St.odch*				
Bayesovský klasifikátor	náhodný výběr	0,908	*0,059*	14,90	*8,39*	0,500	0,008	0,296	00:00:14
	BIF*	0,948	*0,004*	27,15	*4,09*	0,927	0,244	0,862	00:04:57
	SFS*	0,963	*0,003*	11,95	*5,30*	0,506	0,181	0,332	01.02.04
	SFFS*	0,969	*0,003*	12,17	*4,66*	0,556	0,259	0,387	09:13:03
	DOS	0,973	*0,002*	8,85	*2,36*	0,584	0,419	0,429	12:49:59
3NN	náhodný výběr	0,935	*0,061*	14,9	*8,30*	0,501	0,009	0,297	00:00:45
	BIF*	0,970	*0,002*	24,78	*3,70*	0,912	0,513	0,840	00:38:39
	SFS*	0,976	*0,002*	15,45	*5,74*	0,584	0,148	0,401	07:27:39
	SFFS*	0,979	*0,002*	17,96	*5,67*	0,658	0,149	0,481	33:53:55
	DOS	0,980	*0,001*	13,27	*4,25*	0,565	0,227	0,393	116:47
SVM	náhodný výběr	0,942	*0,059*	14,94	*8,58*	0,502	0,008	0,295	00:00:50
	BIF*	0,974	*0,003*	21,67	*2,71*	0,929	0,774	0,875	01:01:48
	SFS*	0,982	*0,002*	9,32	*4,12*	0,433	0,185	0,283	07:13:02
	SFFS*	0,983	*0,002*	10,82	*4,58*	0,472	0,179	0,310	30:28:02
	DOS	0,985	*0,001*	8,70	*3,42*	0,442	0,222	0,295	74:28:51

Velmi malé hodnoty CW_{rel} mohou indikovat hlubší problém daného procesu výběru příznaků. Je možné, že v dané úloze nejsou příznaky dostatečně rozlišitelné (a metody výběru příznaků tak generují více či méně náhodné podmnožiny), popř. dochází k výraznému přeučení (přílišnému přizpůsobení speciálním vlastnostem momentálně navzorkované trénovací datové podmnožiny). Dále poznamenejme, že malé hodnoty stability jsou často doprovázeny většími odchylkami ve velikosti podmnožin.

13.10 Shrnutí a další vývoj

Tato kapitola byla věnována problému výběru příznaků ve statistickém rozpoznávání. *Výběr příznaků* je jedním z nejčastěji používaných přístupů k redukci dimenzionality, jež je první ze dvou klíčových fází řešení problémů rozpoznávání. Druhou fází je vlastní klasifikace, prováděná na příznakovém podprostoru určeném ve fázi první. Výhodou *výběru příznaků* oproti *extrakci* je zachování významu původních měření (např. v medicíně je výhodné určit, která potenciálně drahá měření je možné pro účely automatické diagnostiky vynechat). Podali jsme zejména přehled dostupných vyhledávacích strategií a kritické srovnání jejich vlastností výhod i omezení zejména ve smyslu použitelnosti pro výběr příznaků. Dále jsme navrhli několik přehledových dělení známých metod na základě různých pohledů na cíle či průběh řešení problému. Současný stav problematiky můžeme shrnout takto:

Optimální metody výběru příznaků přicházejí v úvahu pouze pro problémy nepříliš velké dimenzionality. Není-li časově únosné provést úplné prohledání, potřebujeme navíc kritérium ohodnocení podmnožin splňující podmínku monotónnosti, aby bylo možné použít některý z algoritmů větví a mezí (Branch & Bound). Ačkoliv nejmodernější varianty tohoto algoritmu běží až o dva řády rychleji než varianta základní, zůstává řada principiálních omezení. Skutečná výkonnost algoritmů větví a mezí může při nepříznivých vlastnostech dat a kriteriální funkce vést k zdlouhavému výpočtu ne nutně rychlejšímu, než by bylo úplné prohledání. Běh těchto algoritmů má v principu exponenciální charakter a pro konkrétní problém není doposud předem možné odhadnout očekávanou dobu výpočtu. Ačkoliv pro některé problémy algoritmus nachází řešení velmi rychle,

reálně je nutné počítat s omezením na problémy dimenzionality do asi 40 až 50 při obvyklé době výpočtu hodin až dnů. Za nejmodernější algoritmy větví a mezí lze považovat algoritmy využívající samoučícího predikčního mechanismu. Jsou jimi tzv. *rychlý algoritmus větví a mezí FBB* (odst. 13.4.4) a *algoritmus větví a mezí s částečnou predikcí BBPP* (odst. 13.4.3).

Suboptimální metody jsou zřejmě nejčastěji používanými metodami výběru příznaků. Ačkoliv nezaručují nalezení optima vzhledem k zvolenému kritériu, často toto optimum nacházejí, nebo se k němu značně přibližují. Na rozdíl od optimálních metod bývají suboptimální metody tolerantní k nemonotónnímu chování kritéria a jsou proto použitelné pro přímou maximalizaci úspěšnosti klasifikátorů. Časová náročnost suboptimálních metod je obvykle polynomická na rozdíl od exponenciální náročnosti metod optimálních. Úpravou parametrů lze často volit mezi délkou výpočtu a očekávanou blízkostí řešení k optimu. Základní suboptimální metody jsou velmi jednoduché a snadno implementovatelné. Tradiční metody lze obvykle rozdělit na metody zdola–nahoru a shora–dolů podle převažujícího vývoje kardinality momentálně prohledávaných podmnožin. Nejpokročilejšími metodami této řady metod jsou hojně citované *plovoucí metody*, které v jednom průchodu nachází řešení ve všech dimenzích a obvykle výrazně překonávají metody dosavadní. Ještě širší možnosti nabízejí *oscilační metody*, které na rozdíl od ostatních metod zpracovávají v každém kroku podmnožiny požadované kardinality (odst. 13.5.6). Algoritmus *dynamického oscilačního vyhledávání* navíc optimalizuje velikost podmnožiny příznaků. Alternativně lze k výběru příznaků využít i různých randomizovaných algoritmů, např. algoritmů genetických (Mayer a kol., 2000) atd.

Koncepčně odlišné jsou metody výběru příznaků založené na modelování struktury dat za použití směsi hustot pravděpodobnosti speciálního typu. Tento přístup je vhodný pro situace, když máme k dispozici rozsáhlá trénovací data, o jejichž vlastnostech ale nic nevíme a hrozí tedy, že jde o data obtížně modelovatelná, např. multimodální. Jmenujme dvě varianty tohoto přístupu, tzv. *aproximační metodu*, vhodnou k reprezentaci dat, a *divergenční metodu*, vhodnou k separaci tříd. V obou případech výpočet zahrnuje konvergující iterativní výpočty parametrů směsi, přičemž výběr příznaků je integrální součástí výpočtu v každé iteraci.

Oblast redukce dimenzionality poskytuje široké možnosti dalšího vývoje. Je možné věnovat úsilí urychlování doposud příliš pomalých optimálních metod. Na druhé straně lze uvažovat o sofistikovanějších hledacích schématech zvyšujících šance, že nalezneme optima u metod suboptimálních. Neméně důležité je ovšem zkoumat *vlastnosti kriteriálních funkcí* a dále *klasifikátorů*, v jejichž lepší výkonnost a přesnost má redukce dimenzionality vyústit.

Při použití existujících nástrojů je navíc nutné brát v úvahu řadu potenciálních problémů. Podobně jako při konstrukci klasifikátorů musíme při výběru příznaků zabránit *přetrénování*, neboli přílišnému přizpůsobení se výsledku trénovací množiny (Raudys, 2006). Aktuálním tématem je též zkoumání *stability metod výběru příznaků*. Přetrénování, malá stabilita, popř. další komplikace, mohou – nejsou-li včas rozeznány – zásadně negativně ovlivnit přesnost výsledného rozhodovacího pravidla na nových datech. Zejména u vysokodimenzionálních problémů (kategorizace textů, vyhledávání genů, zpracování obrazových dat na úrovni pixelů) jsou obvykle používány pouze nejjednodušší metody výběru příznaků, a to nejen pro výpočetní složitost, ale také kvůli přílišnému nebezpečí přetrénování. Aktuální výsledky však naznačují praktickou možnost netriviálního výběru příznaků i v takto obtížných problémech (Somol a kol., 2011).

Množství metod a nástrojů je dnes již tak rozsáhlé, že je vhodné uvažovat o vytvoření podpůrných systémů usnadňujících uživateli orientaci a volbu postupu v konkrétních případech. První kroky byly učiněny v podobě vytvoření souborných knihoven relevantních metod (van der Heijden a kol., 2004), popř. kompaktnějších softwarových nástrojů (Somol a kol., 2002), (Pudil a kol., 2002). Tvorba přehledného sjednocujícího systému zůstává jedním ze směrů práce také našeho týmu. Pro získání aktuálních informací a volně dostupné rozsáhlé softwarové knihovny Feature Selection Toolbox odkazujeme na portál http://fst.utia.cz.

Literatura

Arauzo-Azofra A., Benítez J. M., Castro J. L.: C-focus:A continuous extension of focus. In: *Proc. 7th Online World Conf. on Soft Computing in Industrial Applications*, 2003, s. 225–232.

Asuncion A., Newman D.: UCI machine learning repository, 2007, http://www.ics.uci.edu/ mlearn/ mlrepository.html.

Boekee D. E., van der Lubbe J. C. A.: Some aspects of error bounds in feature selection, *Pattern Recognition*, 11, 1979, s. 353–360.

Bouguila N.: On multivariate binary data clustering and feature weighting, *Computational Statistics and Data Analysis*, 54, 2010, s. 120–134.

Bouguila N., Guebaly W. E.: Discrete data clustering using finite mixture models, *Pattern Recognition*, 42(1), 2009, s. 33–42.

Caruana R., Freitag D.: Greedy attribute selection. In: *Int. Conf. on Machine Learning ML–94*. Morgan Kaufmann, 1994, s. 28–36.

Das S.: Filters wrappers and a boosting-based hybrid for feature selection. In: *ICML '01: Proc. 18th Int. Conf. on Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisko 2001, s. 74–81.

Dasarathy B. V. (ed.): Nearest Neighbor Pattern Classification Techniques, *IEEE Computer Society Press*, 1991.

Dash M., Choi K., Scheuermann P., Liu H.: Feature selection for clustering – a filter solution. In: *ICDM '02: Proc. 2002 IEEE Int. Conf. on Data Mining*, volume 00. IEEE Computer Society, Washington 2002, 115 s.

Debuse J. C. W., Rayward-Smith V. J.: Feature subset selection within a simulated annealing data mining algorithm, *J. Intell. Inf. Syst.*, 9(1), 1997, s. 57–81.

Devijver P. A., Kittler J.: *Pattern Recognition: A Statistical Approach*. Prentice Hall, Englewood Cliffs, Londýn 1982.

Duda R. O., Hart P. E., Stork D. G.: *Pattern Classification*. (2. vyd.), Wiley-Interscience, 2000.

Dunne K., Cunningham P., Azuaje F.: *Solutions to Instability Problems with equential Wrapper based Approaches to Feature Selection*. Technical Report TCD-CS-2002-28, Department of Computer Science, Trinity College, Dublin 2002.

Foroutan I., Sklansky J.: Feature selection for automatic classification of nongaussian data, *IEEE Trans. on Systems Man and Cybernetics*, 17, 1987, s. 187–198.

Fukunaga K.: *Introduction to Statistical Pattern Recognition*. (2. vyd.), Academic Press Professional, Inc., San Diego 1990.

Gengler M., Coray G.: Aparallel best-first Branch&Bound algorithm and its axiomatization, *Parallel Algorithms Appl*, 2(1–2), 1994, s. 61–80.

Graham M. W., Miller D. J.: Unsupervised learning of parsimonious mixtures on large spaces with integrated feature and component selection, *IEEE Trans. on Signal Processing*, 54(4), 2006, s. 1289–1303.

Guyon I., Elisseeff A.: An introduction to variable and feature selection, *J. Mach. Learn. Res.*, 3, 2003, s. 1157–1182.

Hamamoto Y., Uchimura S., Matsuura Y., Kanaoka T., Tomita S.: Evaluation of the branch and bound algorithm for feature selection, *Pattern Recognition Letters*, 11(7), 1990, s. 453–456.

Hastie T., Tibshirani R.: Discriminant analysis by gaussian mixtures, *Journal of the Royal Statistical Society*, 58, 1996, s. 155–176.

Heijden van der F., Duin R. P., de Ridder D., Tax D. M.: *Classification Parameter Estimation and State Estimation: An Engineering Approach Using MATLAB*. Wiley, New York 2004.

Hussein F., Ward R., Kharma N.: Genetic algorithms for feature selection and weighting a review and study, *icdar* 00, 2001, 1240 s.

Chaikla N., Qi Y.: Genetic algorithms in feature selection. In: *IEEE Int. Conf. on Systems Man and Cybernetics*. vol. 5, IEEE Computer Society, Washington 1999, s. 538–540.

Chang C.-C., Lin C.-J.: *LIBSVM: a library for SVM*, 2001, http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chen X. W.: An improved branch and bound algorithm for feature selection, *Pattern Recognition Letters*, 24(12), 2003, s. 1925–1933.

Iamnitchi A., Foster I.: A problem-specific fault-tolerance mechanism for asynchronous distributed systems. In: *2000 International Conference on Parallel Processing (ICPP'00)*. IEEE, Washington, Brusel, Tokio 2000, s. 4–14.

Jain A., Zongker D.: Feature selection: Evaluation application and small sample performance, *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(2), 1997, s. 153–158.

Jensen R.: *Performing Feature Selection with ACO*, vol. 34 of *Studies in Computational Intelligence*. Springer, Berlín, Heidelberg 2006, s. 45–73.

Kalousis A., Prados J., Hilario M.: Stability of feature selection algorithms: A study on high-dimensional spaces. *Knowledge and Information Systems*, 12(1), 2007, s. 95–116.

Kirkpatrick S., Gelatt C. D., Vecchi M. P.: Optimization by simulated annealing, *Science*, 220, 1983, s. 671–680.

Kohavi R., John G. H.: Wrappers for feature subset selection, *Artif. Intell.*, 97(1–2), 1997, s. 273–324.

Koller D., Sahami M.: Toward optimal feature selection. In: *13th Int. Conf. on Machine Learning*, 1996, s. 284–292.

Kononenko I.: Estimating attributes: Analysis and extensions of relief. In: *ECML-94: Proc. European Conf. on Machine Learning*. Springer-Verlag, New York, Inc., Secaucus, NJ, 1994, s. 171–182.

Korf K. E.: Artificial intelligence search algorithms. In: *Algorithms and Theory of Computation Handbook*. CRC Press, 1999.

Krishnapuram B., Hartemink A. J.: A bayesian approach to joint feature selection and classifier design, *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(9), 2004, s. 1105–1111.

Křížek P., Kittler J., Hlaváč V.: Improving stability of feature selection methods. In: *Proc. 12th Int. Conf. on Computer Analysis of Images and Patterns*, vol. LNCS 4673, Springer-Verlag, Berlín, Heidelberg 2007, s. 929–936.

Kudo M., Sklansky J.: Comparison of algorithms that select features for pattern classifiers, *Pattern Recognition*, 33(1), 2000, s. 25–41.

Kumar V., Kanal L. N.: A general branch and bound formulation for understanding and synthesizing and/or tree search procedures, *Artificial Intelligence*, 21(1–2), 1983, s. 179–198.

Kuncheva L. I.: A stability index for feature selection. In: *Proc. 25th IASTED International Multi-Conference AIAP'07*. ACTA Press, Anaheim 2007, s. 390–395.

Law M. H. C., Figueiredo M. A. T., Jain A.: Simultaneous feature selection and clustering using mixture models, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(9), 2004, 1154–1166.

Lawler E. L., Wood D. E.: Branch-and-bound methods: A survey, *Operations Research*, 14, 1966, s. 699–719.

Liu H., Setiono R.: Scalable feature selection for large sized databases. In: *Proceedings of the Fourth World Congress on Expert Systems*. Morgan Kaufmann, 1998, s. 521–528.

Liu H., Yu L.: Toward integrating feature selection algorithms for classification and clustering, *IEEE Transactions on Knowledge and Data Engineering*, 17(4), 2005, s. 491–502.

Mayer H. A., Somol P., Huber R., Pudil P.: Improving statistical measures of feature subsets by conventional and evolutionary approaches. In: *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*. Springer-Verlag, Londýn 2000, s. 77–86.

McKenzie P., Alder M.: Initializing the EM algorithm for use in Gaussian mixture modelling, *Machine Intelligence and Pattern Recognition*, 16, 1994, s. 91.

McLachlan G. J.: *Discriminant analysis and statistical pattern recognition*. Wiley-IEEE, 2004.

Miller D. J., Browning J.: A mixture model and EM-based algorithm for class discovery robust classification and outlier rejection in mixed labeled/unlabeled data sets, *IEEE Trans. Pattern Analysis and Machine Intelligence*, 25(11), 2003, s. 1468–1483.

Mitschele-Thiel A.: Integrated system development with the DSPL programming environment. In: Joubert G. R., Trystram D., Peters F. J., Evans D. J. (eds): *Parallel Computing: Trends and Applications PARCO'93*. Elsevier, Grenoble 1994, s. 635–638.

Nakariyakul S., Casasent D. P.: Adaptive branch and bound algorithm for selecting optimal features, *Pattern Recogn. Lett.*, 28(12), 2007, s. 1415–1427.

Nakariyakul S. & Casasent D. P.: An improvement on floating search algorithms for feature subset selection, *Pattern Recognition*, 42(9), 2009, s. 1932–1940.

Narendra P. M., Fukunaga K.: A branch and bound algorithm for feature subset selection, *IEEE Trans. Computers*, 26(9), 1977, s. 917–922.

Nilsson N.: *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, New York 1971.

Nilsson N. J.: *Artificial Intelligence: A New Synthesis*. Morgan Kaufmann Publishers, San Francisko 1998.

Novovičová J., Pudil P., Kittler J.: Divergence based feature selection for multimodal class densities, *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(2), 1996, s. 218–223.

Novovičová J., Somol P., Pudil P.: Oscillating feature subset search algorithm for text categorization. In: *Structural Syntactic and Statistical Pattern Recognition*, vol. LNCS 4109, Springer-Verlag, Berlín, Heidelberg 2006, s. 578–587.

Novovičová J., Pudil P.: Feature selection and classification by modified model with latent structure. In: Karný M., Warwick K., Kurková V. (eds): *Dealing with Complexity: A Neural Networks Approach*. Springer Verlag, 1998, s. 126–140.

Palm H. C.: A new method for generating statistical classifiers assuming linear mixtures of gaussian densities. In: *ICPR*, 1994, s. B:483–486.

Pudil P., Novovičová J., Choakjarernwanit N., Kittler J.: Feature selection based on the approximation of class densities by finite mixtures of special type, *Pattern Recognition*, 28(9), 1995, s. 1389–1398.

Pudil P., Novovičová J., Kittler J.: Floating search methods in feature selection, *Pattern Recognition Letters*, 15(11), 1994a, s. 1119–1125.

Pudil P., Novovičová J., Somol P.: Feature selection toolbox software package, *Pattern Recognition Letters*, 23(4), 2002, s. 487–492.

Pudil P., Novovičová J., Kittler J.: Simultaneous learning of decision rules and important attributes for classification problems in image analysis, *Image and Vision Computing*, 12, 1994b, s. 193–198.

Raudys Š. J.: Feature over-selection. In: *Structural Syntactic and Statistical Pattern Recognition*, vol.LNCS 4109, Springer-Verlag, Berlín, Heidelberg 2006, s. 622–631.

Redner R. A., Walker H. F.: Mixture densities maximum likelihood and the EM algorithm, *SIAM Review*, 26(2), 1984, s. 195–239.

Reunanen J.: Overfitting in making comparisons between variable selection methods, *J. Mach. Learn. Res.*, 3, 2003, s. 1371–1382.

Ripley B. D. (ed.): *Pattern Recognition and Neural Networks*. 8. vyd., Cambridge University Press, 2005.

Saeys Y., Inaki I. I., Larranga P. L.: A review of feature selection techniques in bioinformatics, *Bioinformatics*, 23(19), 2007, s. 2507–2517.

Sebastiani F. Machine learning in automated text categorization, *ACM Computing Surveys*, 34(1), 2002, s. 1–47.

Sebban M., Nock R.: A hybrid filter/wrapper approach of feature selection using information theory, *Pattern Recognition*, 35, 2002, s. 835–846.

Skurichina M.: *Stabilizing Weak Classifiers*. PhD thesis, Pattern Recognition Group, Delft University of Technology, Netherlands, 2001.

Somol P., Novovičová J., Pudil P.: Flexible-hybrid sequential floating search in statistical feature selection. In: *Structural Syntactic and Statistical Pattern Recognition*, vol. LNCS 4109, Springer-Verlag, Berlín, Heidelberg 2006, s. 632–639.

Somol P. & Novovičová J. (2008a). Evaluating the stability of feature selectors that optimize feature subset cardinality. In: *Structural Syntactic and Statistical Pattern Recognition* vol. LNCS 5342, s. 956–966.

Somol P., Novovičová J., Grim J., Pudil P.: Dynamic oscillating search algorithms for feature selection. In: *ICPR 2008*. IEEE Computer Society, Los Alamitos, CA, 2008b.

Somol P., Pudil P.: Oscillating search algorithms for feature selection. In: *ICPR 2000*, vol. 02. IEEE Computer Society, Los Alamitos, CA, 2000, s. 406–409.

Somol P., Pudil P.: Feature selection toolbox, *Pattern Recognition*, 35(12), 2002, s. 2749–2759.

Somol P., Pudil P., Kittler J.: Fast branch&bound algorithms for optimal feature selection, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(7), 2004, s. 900–912.

Somol P., Pudil P., Novovičová J., & Paclík P.: Adaptive floating search methods in feature selection, *Pattern Recognition Letters*, 20(11–13), 1999, s. 1157–1163.

Somol P., Grim J., Pudil P.: Fast Dependency-Aware Feature Selection in Very-High-Dimensional Pattern Recognition. In: *IEEE SMC*, 2011, s. 502–509.

Sun Y.: Iterative relief for feature weighting: Algorithms theories and applications, *IEEE Trans. Pattern Anal. Mach. Intell*, 29(6), 2007, s. 1035–1051.

Theodoridis S., Koutroumbas K.: *Pattern Recognition*. 3. vyd., Academic Press, 2006.

Webb A. R.: *Statistical Pattern Recognition*. 2. vyd., John Wiley and Sons Ltd., 2002.

Webb G. I.: OPUS: An efficient admissible algorithm for unordered search, *Journal of Artificial Intelligence Research*, 3, 1995, s. 431–465.

Whitney A. W.: A direct method of nonparametric measurement selection, *IEEE Trans. Comput.*, 20(9), 1971 s. 1100–1103.

Xing E. P.: *Feature Selection in Microarray Analysis*. Springer, 2003, s. 110–129.

Xu C., Tschöke S., Monien B.: Performance evaluation of load distribution strategies in parallel branch and bound computations. In: *Proceedings of the 7th IEEE Symposium on Parallel and Distributed Processing SPDP'95*, 1995, s. 402–405.

Yang M., Das C.: A parallel optimal branch-and-bound algorithm for min-based multiprocessors. In: *Proceedings of the IEEE Int. Conf. on Parallel Processing*, 1999, s. 112–119.

Yang Y., Pedersen J. O.: A comparative study on feature selection in text categorization. In: *ICML '97: Proc. 14th Int. Conf. on Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisko 1997, s. 412–420.

Yu B., Yuan B. Z.: A more efficient branch and bound algorithm for feature selection, *Pattern Recognition*, 26(6), 1993, s. 883–889.

Yu L., Liu H.: Feature selection for high-dimensional data: A fast correlation based filter solution. In: *ICML-03: Proc. 20th Int. Conf. on Machine Learning*, vol. 20, Morgan Kaufmann Publishers Inc., Washington 2003, s. 856–863.

Zhang H., Sun G.: Feature selection using tabu search method, *Pattern Recognition*, 35, 2002, s. 701–711.

Česko–anglický slovník

4ft-kvantifikátor	4ft-quantifier
4ft-kvantifikátor dvojité fundované implikace	4ft-quantifier of double founded implication
4ft-kvantifikátor fundované implikace	4ft-quantifier of founded implication
4ft-kvantifikátor nadprůměrné závislosti	above average 4ft-quantifier

A

agent dodávky	supply agent
agent dopravy	transportation agent
agent objednávky	order agent
agent výrobku	product agent
agent výrobní stanice	workstation agent
aktivační funkce neuronu	activation function of neuron
aktivní učení	active learning
algoritmus Forward	Forward algorithm
algoritmus Forward–Backward	Forward-Backward algorithm
algoritmus větví a mezí	branch & bound algorithm
analogie	analogy
analýza nákupního košíku	market basket analysis
antecedent	antecedent
antijednota	anti-unity
antirigidní vlastnost	anti-rigid property
architektura neuronové sítě	neural network architecture
asociační pravidlo	association rule
asynchronní výpočet neuronové sítě	asynchronous computation of neural network
atribut	attribute, feature
automatické dokazování vět	automated theorem proving
axiom bezespornosti	consistency axiom
axiom distributivity	distribution axiom
axiom znalosti	knowledge axiom
axiomatický systém	axiomatic system

B

Baumův–Welchův algoritmus	Baum–Welch algorithm
Bhattacharyova vzdálenost	Bhattacharya distance
binární rezoluce	binary resolution
booleovský dotaz	boolean query
boosting	boosting
bootstrapping	bootstrapping

C

celková délka propojení obvodu	total wire length of circuit
CS-strom	conflict set tree (CS-tree)

D

datová a objektová vlastnost	datatype and object property
dedukční pravidla pro asociační pravidla	deduction rules for association rules
deskripční logika	description logic
deskriptivní logika	descriptive logic
diagnóza	diagnosis
dokazatelnost	provability
doménová ontologie	domain ontology
domněnka	conjecture
dopředná neuronová síť	feedforward neural network
dotazovací jazyk	query language
důkaz formule	proof of a formula

E

empirické učení	empirical learning
energetická funkce	energy function
esenciální vlastnost	essential property
excitace neuronu	excitation of neuron
extrakce informací	information extraction
extrakční ontologie	extraction ontology

F

formalizace problému	problem formalization
formální důkaz	formal proof

G

glosář	glossary
graf zúplnění	completion graph

H

Heavisideova funkce	Heaviside function
hledání řízené ontologií	ontology driven search
hledání ve strukturovaných datech	search in structured data
hloubka sítě	depth of network
holičův paradox	barber's Paradox
Hopfieldova neuronová síť	Hopfield neural network
Hopfieldův jazyk	Hopfield language
hybridní metody	hybrid methods

I

identita	identity
individuál	individual

individuálně nejlepší příznaky	individually best features
individuum	individual
inkrementální učení	incremental learning
integrované metody	embedded methods
integrování znalostí	knowledge integration

J

jazyk výrokové modální logiky	language of propositional modal logic
jednota	unity

K

kategorie	category
klasifikace	classification
klauzální dokazování	clausal theorem proving
klauzifikátor	clausifier
kognitivní agent	cognitive agent
koncept	concept
konceptualizace	conceptualization
konceptuální graf	conceptual graph
konjunktivní dotaz	conjunctive query
konvence jmenná	naming convention
korektnost axiomatického systému	soundness of axiomatic system
Kripkeho formule	Kripke formula
Kripkeho sémantika	Kripke semantics
Kripkeho struktura	Kripke structure
kritérium informativnosti	criterion of informativeness
kvantifikátor	quantifier

L

lineární prahová funkce	linear threshold function
lineární temporální logika	linear temporal logic
Ljapunovova funkce	Lyapunov function
logicko-strukturní návrhový vzor	logical-structural design pattern
logický důsledek	logical consequence
logika dokazatelnosti	provability logic
logistická sigmoida	logistic sigmoid

M

mapování ontologií	ontology mapping
materiální implikace	material implication
Markovův model s maximální entropií	maximal entropy Markov model
matice dat	data matrix
metodika eXtreme Design	eXtreme Design methodology
metody vysvětlování modelovacích chyb black-box	black-box methods for error explanation

metody vysvětlování modelovacích chyb glass-box	glass-box methods for error explanation
množinová sémantika	set-theoretic semantics
modalita	modality
model interpretace	interpretation model
modus ponens	modus ponens
multiagentní systém	multiagent system

N

negativní introspekce	negative introsepction
negativní normální forma	negative normal form
nekonečná posloupnost obvodů	infinite family of circuits
neorientovaný grafický model	undirected graphical model
nerigidní vlastnost	non-rigid property
nerozlišená proměnná	undistinguished variable
neúplná informace	missing information
neuronová síť	neural network
neuronová síť pracující ve spojitém čase	continuous-time neural network
neuronový akceptor jazyka	neural language acceptor
normalizace taxonomie	taxonomy normalization

O

obraz	pattern
obsahový návrhový vzor	content ontology design pattern
observační kalkuly	observational calculi
observační kalkuly asociačních pravidel	observational calculi of association rules
obvod AND–OR	AND-OR circuit
odvozený booleovský atribut	derived Boolean attribute
odvozovací pravidlo	deduction rule
ontologický závazek	ontological commitment
ontologie	ontology
optimální metoda	optimal method
orientovaný grafický model	directed graphical model
oscilační vyhledávání	oscillating search
ověřovač formálních důkazů	formal proof checkers
ověřování konzistence	consistency checking

P

paralelní režim neuronové sítě	parallel mode of neural network
páteřní ontologie	backbone ontology
platnost formule	formula validity
plovoucí vyhledávání	floating search
podmíněná náhodná pole	conditional random fields
podmínka identity	identity condition
podmínka monotónnosti	monotonicity condition

podmnožinové blokování	subset blocking
posun konceptu	concept drift
pouzdro	wrapper
pozitivní introspekce	positive introspection
prahové hradlo	threshold gate
prahový obvod	threshold circuit
pravděpodobnostní grafický model	probabilistic graphical model
pravděpodobnostní neuronová síť	probabilistic neural network
pravdivost ve struktuře	validity in a structure
pravidlo faktorizace	factorization rule
pravidlo generalizace	knowledge generalization rule
pravidlo řezu	cut rule
procedura	procedure
procedura 4ft-Miner	4ft-Miner procedure
procedura ASSOC	ASSOC procedure
propojená data	linked data
protokol Plan-Commit-Execute	Plan-Commit-Execute protocol
prvotní formule	primitive formula
předpoklad	premise
předpoklad otevřeného světa	open world assumption
přenášení vlastnosti	carrying property
přesnost extrakce	extraction precision
přesnost Vilainova	Vilain precision
přeučení	overfitting
přímý spor	direct clash
příznak	feature

R

rámcový model	frame model
redukce dimenzionality	dimensionality reduction
Reiterův algoritmus	Reiter algorithm
rekurentní neuronová síť	recurrent neural network
relace *n*-ární	*n*-ary relation
relace přípustnosti	possibility relation
reprezentace dat	data representation
revize znalostí	knowledge revision
rezoluční metoda	resolution method
rezolventa	resolvent
rezolvované literály	literals resolved upon
rigidní vlastnost	rigid property
role	role
rozhraní člověk-stroj	human-machine interface
rozlišená proměnná	distinguished variable
rozpoznávání pojmenovaných entit	named entity recognition

S

saturovaná lineární funkce	saturated-linear function

sekvenční mód neuronové sítě	sequential mode of neural network
sekvenční vyhledávání	sequential search
selekce příznaků	feature selection
sémantická anotace	semantic annotation
sémantická síť	semantic network
sémantické tableau	semantic tableau
sémantický důsledek	semantic consequence
sémantický web	semantic web
sémantika možných světů	possible words semantics
semirozhodnutelná logika	semidecidable logic
simulační systém MAST	MAST simulation system
skrytý Markovův model	hidden Markov model
směsové metody	mixture methods
smyčka ANL	ANL Loop
splnitelná formule	satisfiable formula
společná znalost	common knowledge
stabilita	stability
stabilní stav	stable state
standardní sigmoida	standard sigmoid
stav neuronové sítě	neural network state
striktní implikace	strict implication
suboptimální metoda	sub-optimal method
subsumpce	subsumption
sukcedent	succedent
symetrická booleovská funkce	symmetric Boolean function
symetrická neuronová síť	symmetric neural network
synaptická váha	synaptic weight
synchronní výpočet neuronové sítě	synchronous computation of neural network
systém adaptivní	adaptive system
systém LISp-Miner	LISp-Miner system
systém učící se	learning systém

T

tablový algoritmus	tableau algorithm
tautologie	tautology
taxonomie	taxonomy
temporální logika	temporal logic
temporální operátor	temporal operator
topologie neuronové sítě	neural network topology
transformační pravidlo	transformation rule
třída	class
třída jako hodnota vlastnosti	class as property value
třídy 4ft-kvantifikátorů	classes of 4ft-quantifiers

U

učení a zapomínání	learning and forgetting
učení na základě podobnosti	similarity based learning

učení se dovednostem	skill refinement
učení se znalostem	knowledge acquisition
úplnost axiomatického systému	completness
úplnost extrakce	recall of extraction
uzavřené tableau	closed tableau
uživatelské rozhraní pro hledání	search user interface

V

validní formule	valid formula
vícevrstvá neuronová síť	multi-layered neural network
Vilainova úplnost	Vilain recall
Viterbiho algoritmus	Viterbi algorithm
vrchní ontologie	upper ontology
výběr příznaků	feature selection
výrobní stanice	workstation
výroková konstanta	propositional constant
výroková proměnná	propositional variable
výrokový atom	propositional atom
vzdálenost	distance

Z

zabalovací procedura	rolling-up technique
základní booleovský atribut	basic Boolean attribute
základní ontologie	fundamental ontology
zamítací dokazování	refutational theorem proving
zdroj	resource
znalostní báze	knowledge base

Ž

žebříčkový kód	ladder code
životní cyklus ontologie	ontology life cycle

Anglicko–český slovník

4ft-Miner procedure	procedura 4ft-Miner
4ft-quantifier	4ft-kvantifikátor
4ft-quantifier of double founded implication	4ft-kvantifikátor dvojité fundované implikace
4ft-quantifier of founded implication	4ft-kvantifikátor fundované implikace
A	
above average 4ft-quantifier	4ft-kvantifikátor nadprůměrné závislosti
active learning	aktivní učení
adaptive system	adaptivní systém
analogy	analogie
AND-OR circuit	obvod AND–OR
ANL Loop	smyčka ANL
antecedent	antecedent
anti-rigid property	antirigidní vlastnost
anti-unity	antijednota
association rule	asociační pravidlo
asynchronous computation of neural network	asynchronní výpočet neuronové sítě
attribute	atribut
automated theorem proving	automatické dokazování vět
axiomatic system	axiomatický systém
B	
backbone ontology	páteřní ontologie
barber's Paradox	holičův paradox
basic Boolean attribute	základní booleovský atribut
Baum-Welch algorithm	Baumův-Welchův algoritmus
Bhattacharya distance	Bhattacharyova vzdálenost
binary resolution	binární rezoluce
black-box methods for error explanation	metody vysvětlování modelovacích chyb black-box
boolean query	booleovský dotaz
boosting	boosting
bootstrapping	bootstrapping
branch & bound algorithm	algoritmus větví a mezí
C	
carrying property	přenášení vlastnosti
category	kategorie

class	třída
class as property value	třída jako hodnota vlastnosti
classes of 4ft-quantifiers	třídy 4ft-kvantifikátorů
classification	klasifikace
clausal theorem proving	klauzální dokazování
clausifier	klauzifikátor
closed tableau	uzavřené tableau
cognitive agent	kognitivní agent
common knowledge	společná znalost
completion graph	graf zúplnění
completness	úplnost axiomatického systému
concept	koncept
concept drift	posun konceptu
conceptual graph	konceptuální graf
conceptualization	konceptualizace
conditional random fields	podmíněná náhodná pole
conflict set tree (CS-tree)	CS-strom
conjecture	domněnka
conjunctive query	konjunktivní dotaz
consistency axiom	axiom bezespornosti
consistency checking	ověřování konzistence
content ontology design pattern	obsahový návrhový vzor
continuous-time neural network	neuronová síť pracující ve spojitém čase
criterion of informativeness	kritérium informativnosti
cut rule	pravidlo řezu

D

data matrix	matice dat
data representation	reprezentace dat
datatype and object property	datová a objektová vlastnost
deduction rule	odvozovací pravidlo
deduction rules for association rules	dedukční pravidla pro asociační pravidla
depth of network	hloubka sítě
derived Boolean attribute	odvozený booleovský atribut
description logic	deskripční logika
descriptive logic	deskriptivní logika
diagnosis	diagnóza
dimensionality reduction	redukce dimenzionality
direct clash	přímý spor
directed graphical model	orientovaný grafický model
distance	vzdálenost
distinguished variable	rozlišená proměnná
distribution axiom	axiom distributivity
domain ontology	doménová ontologie

E

embedded methods	integrované metody
empirical learning	empirické učení

energy function	energetická funkce
essential property	esenciální vlastnost
excitation of neuron	excitace neuronu
extraction ontology	extrakční ontologie
extraction precision	přesnost extrakce
eXtreme Design methodology	metodika eXtreme Design

F

factorization rule	pravidlo faktorizace
feature	atribut, příznak
feature selection	selekce příznaků, výběr příznaků
feed-forward neural network	dopředná neuronová síť
floating search	plovoucí vyhledávání
formal proof	formální důkaz
formal proof checkers	ověřovač formálních důkazů
formula validity	platnost formule
Forward algorithm	algoritmus Forward
Forward-Backward algorithm	algoritmus Forward–Backward
frame model	rámcový model
fundamental ontology	základní ontologie

G

glass-box methods for error explanation	metody vysvětlování modelovacích chyb glass-box
glossary	glosář

H

Heaviside function	Heavisideova funkce
hidden Markov model	skrytý Markovův model
Hopfield language	Hopfieldův jazyk
Hopfield neural network	Hopfieldova neuronová síť
human-machine interface	rozhraní člověk–stroj
hybrid methods	hybridní metody

I

identity	identita
identity condition	podmínka identity
incremental learning	inkrementální učení
individual	individuál, individuum
individually best features	individuálně nejlepší příznaky
infinite family of circuits	nekonečná posloupnost obvodů
information extraction	extrakce informací
interpretation model	model interpretace

K

knowledge acquisition	učení se znalostem
knowledge axiom	axiom znalosti

knowledge base	znalostní báze
knowledge generalization rule	pravidlo generalizace
knowledge integration	integrování znalostí
knowledge revision	revize znalostí
Kripke formula	Kripkeho formule
Kripke semantics	Kripkeho sémantika
Kripke structure	Kripkeho struktura

L

ladder code	žebříčkový kód
language of propositional modal logic	jazyk výrokové modální logiky
learning and forgetting	učení a zapomínání
learning system	systém učící se
linear temporal logic	lineární temporální logika
linear threshold function	lineární prahová funkce
linked data	propojená data
literals resolved upon	rezolvované literály
logical consequence	logický důsledek
logical-structural design pattern	logicko-strukturní návrhový vzor
logistic sigmoid	logistická sigmoida
Lyapunov function	Ljapunovova funkce

M

market basket analysis	analýza nákupního košíku
MAST simulation system	simulační systém MAST
material implication	materiální implikace
maximal entropy Markov model	Markovův model s maximální entropií
missing information	neúplná informace
mixture methods	směsové metody
modality	modalita
modus ponens	modus ponens
monotonicity condition	podmínka monotónnosti
multiagent system	multiagentní systém
multi-layered neural network	vícevrstvá neuronová síť

N

named entity recognition	rozpoznávání pojmenovaných entit
naming convention	konvence jmenná
n-ary relation	relace *n*-ární
negative introsepction	negativní introspekce
negative normal form	negativní normální forma
neural language acceptor	neuronový akceptor jazyka
neural network	neuronová síť
neural network architecture	architektura neuronové sítě
neural network state	stav neuronové sítě
neural network topology	topologie neuronové sítě

neuron activation function	aktivační funkce neuronu
non-rigid property	nerigidní vlastnost

O

observational calculi	observační kalkuly
ontological commitment	ontologický závazek
ontology	ontologie
ontology driven search	hledání řízené ontologií
ontology life cycle	životní cyklus ontologie
ontology mapping	mapování ontologií
open world assumption	předpoklad otevřeného světa
optimal method	optimální metoda
order agent	agent objednávky
oscillating search	oscilační vyhledávání
overfitting	přeučení

P

parallel mode of neural network	paralelní režim neuronové sítě
pattern	obraz
Plan-Commit-Execute protocol	protokol Plan-Commit-Execute
positive introspection	pozitivní introspekce
possibility relation	relace přípustnosti
possible words semantics	sémantika možných světů
premise	předpoklad
primitive formula	prvotní formule
probabilistic graphical model	pravděpodobnostní grafický model
probabilistic neural network	pravděpodobnostní neuronová síť
problem formalization	formalizace problému
procedure	procedura
product agent	agent výrobku
proof of a formula	důkaz formule
propositional atom	výrokový atom
propositional constant	výroková konstanta
propositional variable	výroková proměnná
provability	dokazatelnost
provability logic	logika dokazatelnosti

Q

quantifier	kvantifikátor
query language	dotazovací jazyk

R

recall of extraction	úplnost extrakce
recurrent neural network	rekurentní neuronová síť
refutational theorem proving	zamítací dokazování
Reiter algorithm	Reiterův algoritmus

resolution method	rezoluční metoda
resolvent	rezolventa
resource	zdroj
rigid property	rigidní vlastnost
rolling-up technique	zabalovací procedura

S

satisfiable formula	splnitelná formule
saturated-linear function	saturovaná lineární funkce
search in structured data	hledání ve strukturovaných datech
search user interface	uživatelské rozhraní pro hledání
semantic annotation	sémantická anotace
semantic consequence	sémantický důsledek
semantic network	sémantická síť
semantic tableau	sémantické tableau
semantic web	sémantický web
semidecidable logic	semirozhodnutelná logika
sequential mode of neural network	sekvenční mód neuronové sítě
sequential search	sekvenční vyhledávání
set-theoretic semantics	množinová sémantika
similarity based learning	učení na základě podobnosti
skill refinement	učení se dovednostem
soundness of axiomatic system	korektnost axiomatického systému
stability	stabilita
stable state	stabilní stav
standard sigmoid	standardní sigmoida
strict implication	striktní implikace
sub-optimal method	suboptimální metoda
subset blocking	podmnožinové blokování
subsumption	subsumpce
succedent	sukcedent
supply agent	agent dodávky
symmetric Boolean function	symetrická booleovská funkce
symmetric neural network	symetrická neuronová síť
synaptic weight	synaptická váha
synchronous computation of neural network	synchronní výpočet neuronové sítě

T

tableau algorithm	tablový algoritmus
tautology	tautologie
taxonomy	taxonomie
taxonomy normalization	normalizace taxonomie
temporal logic	temporální logika
temporal operator	temporální operátor
threshold circuit	prahový obvod

threshold gate	prahové hradlo
total wire length of circuit	celková délka propojení obvodu
transformation rule	transformační pravidlo
transportation agent	agent dopravy
U	
undirected graphical model	neorientovaný grafický model
undistinguished variable	nerozlišená proměnná
unity	jednota
upper ontology	vrchní ontologie
V	
valid formula	validní formule
validity in a structure	pravdivost ve struktuře
Vilain precision	Vilainova přesnost
Vilain recall	Vilainova úplnost
Viterbi algorithm	Viterbiho algoritmus
W	
workstation	výrobní stanice
workstation agent	agent výrobní stanice
wrapper	pouzdro, wrapper

Rejstřík

4ft-kvantifikátor 351
– dvojité fundované implikace 352
– fundované implikace 352
– nadprůměrné závislosti 352
4ft-kvantifikátory, třídy 380

A
AA-kvantifikátor 352
ABox 127
agent dodávky 189
– dopravy 189
– kognitivní 24
– objednávky 188
– výrobku 188
– výrobní stanice 189
akceptor jazyka neuronový 404
algoritmus Baumův–Welchův 232
– Forward 234
– Forward–Backward 232
– LP^2 225
– Reiterův 143
– tablový 130
– větví a mezí 429
– Viterbiho 233
analogie 344
analýza nákupního košíku 369
antecedent 351
antijednota 62
ATP 249
atribut 350, 424
– booleovský odvozený 350
– – základní 350

B
bagging 337
báze znalostní 96
blokování podmnožinové 137
boosting 337
bootstrapping 220

C
CS-strom 142

D
data propojená 120
diagnóza 144
dokazování klauzální 256
– vět automatické 249
– zamítací 256
domněnka 249
dotaz booleovský 145
– konjunktivní 144
důkaz formální 250
důsledek logický 254

E
extrakce informací 209
extrakce, přesnost 211
– – Vilainova 212
–, úplnost 211
– – Vilainova 212
extrakční ontologie 221

F
filtr 427
FOAF 88
forma normální negativní 271
formalizace problému 255
funkce booleovská symetrická 400
– energetická 407
– Heavisideova 395
– lineární prahová 396
– – saturovaná 395
– Ljapunovova 407

G
glosář 28
graf konceptuální 74
– zúplnění 131

H
hledání řízené ontologií 180
– ve strukturovaných datech 170
hradlo prahové 395

I
identita 60
individuál 127
individuum 98
informace neúplná 365
instance 99
integrování znalostí 336
interpretace, model 129

J
jazyk dotazovací SPARQL-DL 145
– Hopfieldův 410
jednota 62

K
kalkuly observační 377
– – asociačních pravidel 378
kategorie 350
klasifikace 424
klauzifikátor 256
kód žebříčkový 171
koncept 126
konceptualizace 25
konvence jmenná 152
kritérium informativnosti 424

L
LeanTAP 273
literály rezolvované 257
logika deskripční 95
– – ALC 126
– – SHOIN 130
– deskriptivní 77
– semirozhodnutelná 250

M
matice dat 350
metoda GUHA 348
– optimální 426
– rezoluční 256
– suboptimální 444
metodika eXtreme Design 164
metody hybridní 427
– integrované 427
– směsové 457
– vysvětlování modelovacích chyb, black-box 141
– – –, glass-box 141
model grafický neorientovaný 228
– – orientovaný 227
– – pravděpodobnostní 227
– Markovův s maximální entropií 236
– – skrytý 229
– rámcový 73

N
neuron 394
neuron, excitace 395
–, funkce aktivační 395
NNF 271

O
obraz 424
obvod AND–OR 400
– prahový 397
ontologie 21, 87
– doménová 39
– páteřní 65
– pro průmysl 183
– vrchní 38
– základní 38
ontologie, mapování 112
–, životní cyklus 50
ověřovač formálních důkazů 250
ověřování konzistence 130
OWL 95

P
paradox holičův 261
perceptron 395
podmínka identity 60
– monotónnosti 428
pole náhodná podmíněná 238
posloupnost obvodů nekonečná 397
posun konceptu 340
pouzdro 427
pravidlo asociační 351

pravidlo dedukční pro asociační pravidla 382
– faktorizace 261
– řezu 256
procedura 4ft-Miner 349
– ASSOC 350
– GUHA 348
– zabalovací 146
proměnná nerozlišená 145
– rozlišená 145
propojení obvodu, celková délka 401
protokol Plan-Commit-Execute 188
předpoklad 254
– otevřeného světa 103, 155
přenášení vlastnosti 63
přeučení 461
příznak 424
příznaky individuálně nejlepší 445
příznaky, selekce 425
–, výběr 425

R
RDF 88
RDFa 107
RDFS 88
redukce dimenzionality 424
relace *n*-ární 156
reprezentace dat 425
revize znalostí 337
rezoluce binární 256
rezolventa 257
role 126, 156
rozhraní člověk–stroj 172
– uživatelské, pro hledání 178
rozpoznávání pojmenovaných entit 241

S
sémantická anotace 201
sémantika množinová 154
sigmoida logistická 396
síť neuronová 394
– – dopředná (vícevrstvá) 397
– – pracující ve spojitém čase 413
– – pravděpodobnostní 411
– – rekurentní 403
– – symetrická (Hopfieldova) 407
síť neuronová, architektura 394
– –, režim paralelní 404
– –, – sekvenční 403
– –, stav 394
– –, – stabilní 403
– –, topologie 394
– –, výpočet asynchronní 404
– –, – synchronní 404
– sémantická 73
–, hloubka 397
smyčka ANL 257
SPARQL 105
spor přímý 131
stabilita 462
stacking 337
stanice výrobní 188
subsumpce 98
subterminologie minimální, způsobující nesplnitelnost (MUPS) 141
sukcedent 351
synaptická váha 394
systém adaptivní 325
– LISp-Miner 371
– multiagentní 181
– simulační MAST 190
– učící se 325

T
tableau sémantické 269
– uzavřené 272
tautologie 254
taxonomie 98
taxonomie, normalizace 157
TBox 127
TC^0-hierarchie 399
TPTP 262
transformační pravidlo 225
třída 98
– jako hodnota vlastnosti 156

U
učení a zapomínání 341
– aktivní 219
– empirické 328
– inkrementální 331
– na základě podobnosti 330

učení se dovednostem 328
– znalostem 327
URI 88

V
vlastnost antirigidní 59
– datová a objektová 99
– esenciální 59
– nerigidní 59
– rigidní 59
vyhledávání oscilační 448
– plovoucí 447
– sekvenční 445
vzdálenost 427
– Bhattacharyova 427
vzor návrhový logicko-strukturní 156
– – obsahový 158

W
web sémantický 85
wrapper 218

Z
závazek ontologický 33
zdroj 86

Umělá inteligence (6)

prof. Ing. Vladimír Mařík, DrSc., prof. RNDr. Olga Štěpánková, CSc.,
doc. Ing. Jiří Lažanský, CSc.
prof. Ing. Petr Berka, CSc., Ing. Jiří Kléma, Ph.D., Ing. Zdeněk Kouba, CSc.,
Ing. Petr Křemen, Ph.D., Ing. Martin Labský, Ph.D., doc. RNDr. Jana Novovičová, CSc.,
Ing. Marek Obitko, Ph.D., prof. Ing. Pavel Pudil, DrSc., prof. RNDr. Jan Rauch, CSc.,
RNDr. Petr Somol, Ph.D., doc. Ing. Vojtěch Svátek, Ph.D., doc. RNDr. Jiří Šíma, DrSc.,
prof. RNDr. Petr Štěpánek, DrSc., prof. RNDr. Olga Štěpánková, CSc., RNDr. Josef Urban, Ph.D.,
Mgr. Ing. Miroslav Vacura, Ph.D., Ing. Pavel Vrba, Ph.D., RNDr. Jiří Vyskočil, Ph.D.,
Ing. Ondřej Zamazal, Ph.D., doc. Ing. Zdeněk Zdráhal, CSc., doc. Ing. Filip Železný, Ph.D.

Vydalo nakladatelství Academia
Středisko společných činností AV ČR, v. v. i.
Vodičkova 40, 110 00 Praha 1

Vazbu navrhla Jolana Malátková,
pro šestý díl upravil Oleg Man
Redaktor publikace Aleš Bađura
Technické zpracování, sazba, grafická úprava
a zlom Václav Havlíček
Tisk Serifa, s. r. o., Jinonická 80,
150 00 Praha 5

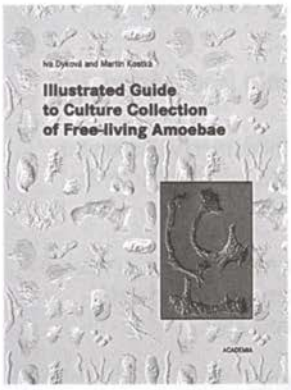
Vydání 1., Praha 2013

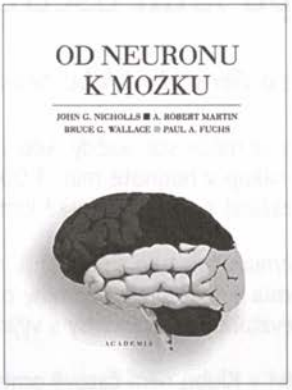
Ediční číslo 10784
ISBN 978-80-200-2276-9
ISBN 80-200-0502-1 (soubor)
ISBN 80-200-0496-3 (1. díl)
ISBN 80-200-0504-8 (2. díl)
ISBN 80-200-0472-6 (3. díl)
ISBN 80-200-1044-0 (4. díl)
ISBN 987-80-200-1470-2 (5. díl)

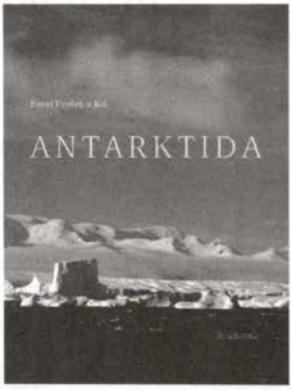
Knihy Nakladatelství Academia zakoupíte také na
www.academiaknihy.cz
www.academiabooks.com
www.eknihy.academia.cz

NAKLADATELSTVÍ ACADEMIA

edice Mimo – přírodní vědy







Vybírejte si knihy v klidu a pohodlí na
www.academiaknihy.cz
a vyzvedněte si je ve Vašem oblíbeném Knihkupectví Academia.

Praha: Václavské nám. 34 – Národní 7 – Na Florenci 3 – Hybernská 8
Brno: nám. Svobody 13; **Ostrava:** Kulturně-literární centrum, Zámecká 2

www.academia.cz
www.eknihy.academia.cz

Naše edice a ediční řady

21. století – Atlasy a Průvodce – Campanula – Europa – Filmová řada – Galileo Gerstner – Historie – Humanitní vědy – Janus – Judaica – Krásná literatura Literární řada – Možné světy – Novověk – Orient – Paměť – Průhledy Přírodní vědy – Psychologie – Stíny – Šťastné zítřky – Umění Vizuální studia – Žena a věda

KLUB ACADEMIA

Staňte se členem
a při nákupu knih ušetřete až 20 %

Podmínky a informace o členství v Klubu Academia jsou následující:

Členem Klubu Academia se může stát každý, kdo v Knihkupectví Academia uskuteční jednorázový nákup v hodnotě min. 1 000 Kč, předloží účtenku a zažádá o vydání členské karty.

Členská karta Klubu Academia poskytuje slevu 20 % na nákup knih z produkce Nakladatelství Academia a 10% slevu na knihy ostatních nakladatelů. Sleva se nevztahuje na časopisy s výjimkou Živy.

Členství v Klubu není časově omezené.

Členská karta je přenosná a pro její vydání není potřeba uvádět osobní údaje.

Slevy lze uplatnit pouze v Knihkupectví Academia, a to v Praze, Brně a Ostravě. Slevy nelze kombinovat a nelze je uplatnit na již zlevněné zboží.

KNIHKUPECTVÍ ACADEMIA

Praha:
Václavské nám. 34
Národní 7
Na Florenci 3
Hybernská 8

Brno:
nám. Svobody 13

Ostrava:
Kulturně-literární centrum
Zámecká 2

www.academia.cz
www.academiaknihy.cz
www.eknihy.academia.cz

Šestý díl knihy „Umělá inteligence“, který se vám dostává do rukou, je soustředěn na problematiku rozpoznávání a automatického zpracování znalostí a představuje techniky, které hrají při řešení těchto úloh zásadní roli. I tento díl pokračuje v tradici předchozích monografií o umělé inteligenci, když začíná filosofickými úvahami a přes stále techničtěji orientované kapitoly dochází autoři publikace až ke kapitolám s vyloženě aplikační tematikou, v tomto případě až k popisu technických systémů agentně orientovaného automatického řízení, pro který má velký význam právě sémantická informace. Kniha, jež je prvním uceleným česky psaným textem o této problematice, nabízí blok kapitol věnovaný ontologiím, sémantickému webu a jeho nástrojům. Na tuto část navazují kapitoly věnované automatizaci logického dokazování a způsobům reprezentace, získávání i využívání znalostí při dialogu více agentů. Neméně zajímavé jsou i kapitoly, které rozebírají aktuální problémy zkoumané v oblasti rozpoznávání i způsoby jejich řešení. Zvláštní pozornost je věnována též výpočetním možnostem neuronových sítí i výběru příznaků v oblasti statistického rozhodování.

ISBN 978-80-200-2276-9

9 788020 022769