Reprezentácia znalostí a riešenie úloh

Logické prístupy

Marián Mach

Košice

2016

doc. Ing. Marián Mach, CSc.
Katedra kybernetiky a umelej inteligencie
Fakulta elektrotechniky a informatiky
Technická univerzita v Košiciach
Marian.Mach@tuke.sk

Lektorovali: *doc. Ing. Kristína Machová, PhD., FEI, TU, Košice*
Ing. Tomáš Cádrik, IT-Impulse s.r.o., Košice
Ing. Martin Čertický, FEI, TU, Košice

ISBN 978-80-553-2632-0

Predslov

Logika a jej využitie pre reprezentáciu znalostí a ich následné aplikovanie pre riešenie úloh patrí do jadra symbolickej vetvy umelej inteligencie už od počiatku existencie tohto vedného odboru. Avšak až v súvislosti s rozvojom výkonu dnešnej výpočtovej techniky jej využitie prekračuje úroveň teoretického nástroja a dostáva sa na úroveň praktického nástroja, vhodného pre riešenie úloh reálnej praxe, schopného konkurovať aktuálne používaným metódam riešenia.

Je možné nájsť mnoho zdrojov v slovenskom jazyku, ktoré sa venujú logike – avšak z matematického pohľadu. Hľadisko riešenia praktických úloh (napr. úloh návrhu, rozvrhovania či plánovania) pomocou logických reprezentácií zvyčajne absentuje. Pre neskúseného čitateľa je pritom veľmi obtiažne urobiť myšlienkový skok od definície a vlastností logických štruktúr k vhodnému spôsobu ich využitia pre konkrétny typ úlohy.

Túto medzeru sa snaží vyplniť predkladaná vysokoškolská učebnica. Aj keď by svojim obsahom mohla čitateľovi slúžiť ako úvod do danej oblasti, v úsilí zachovať kompaktnosť textu sú niektoré pasáže vysvetľované s predpokladom, že čitateľovi niektoré pojmy nie sú úplne neznáme. Vhodnou štartovacou úrovňou sú znalosti o logike na úrovni stredoškolskej látky, ktorými by mali disponovať všetci tí čitatelia, na ktorých je predkladaná učebnica zameraná – cieľovou skupinou sú študenti vysokoškolských, najmä technických, smerov. To však nevylučuje možnosť, že učebnica môže priniesť poučenie a zábavu aj iným typom čitateľov, zaujímajúcim sa o predkladanú problematiku.

Predkladaná látka je rozdelená do troch základných kapitol, pričom každá z nich je venovaná inému druhu logiky. Formálna štruktúra každej kapitoly je pritom rovnaká, spočívajúca na využití vybraného ilustračného príkladu. Kapitola je uvedená takýmto príkladom, reprezentujúcim ilustračný problém a jeho riešenie. Tento problém je v texte kapitoly využitý pre získanie náhľadu, ako prezentované metódy a prístupy môžu byť použité pre reprezentáciu znalostí o danom probléme a následné riešenie tohto problému.

Z hľadiska rozdelenia obsahu každej kapitoly sa úvodná časť kapitoly zaoberá definíciami pojmov a vlastnosťami toho typu logiky, ktorému je venovaná kapitola. Následne je vysvetlené, ako je možné daný typ logiky využiť pre reprezentáciu znalostí o problémoch. Pokračuje sa prezentáciou metód, umožňujúcich zo znalostí o probléme pomocou inferencie získať hľadané riešenie. Záverečne každá kapitola obsahuje sadu príkladov, umožňujúcich prehĺbenie predkladanej problematiky.

Rôzne typy logík prirodzene zdieľajú niektoré definície, vlastnosti, prístupy a metódy. Pre zachovanie kompaktnosti sú tieto preto vysvetľované pri svojom prvom výskyte a pri ich ďalších výskytoch (napr. v iných kapitolách) sa už predpokladá ich znalosť. Vďaka tomu jednotlivé kapitoly sú na sebe závislé a preto sa doporučuje sekvenčné štúdium obsahu jednotlivých kapitol.

Predkladaná vysokoškolská učebnica vznikla na Katedre kybernetiky a umelej inteligencie Fakulty elektrotechniky a informatiky Technickej univerzity v Košiciach. Autor pri jej písaní vychádzal zo svojich skúseností, získaných počas riešenia rôznych úloh ako aj zabezpečovania výuky predmetov so súvisiacou problematikou. Preto je čiastočne aj dielom mnohých študentov, ktorí prinútili autora premýšľať, ako vykladať danú problematiku.

Na tomto mieste by som chcel vyjadriť poďakovanie recenzentom za prečítanie rukopisu, opravu formálnych aj vecných chýb a za cenné pripomienky a námety.

Košice, október 2016 autor

Obsah

Kapitola 1

Výroková logika

Logický systém je definovaný svojou syntaxou a sémantikou. Jazyk, ktorý umožňuje vyjadrovať vety *výrokovej logiky* sa označuje ako *výrokový počet*. Jeho syntaktické pravidlá určujú, čo môžu jednotlivé výrazy obsahovať a ako môžu byť navzájom kombinované. Gramatika pre definíciu tejto syntaxe je v Tab. 1.1 v tvare BNF (Backus-Naurova Forma). SYNTAX

$<$ veta $>$	::=	$<$ atomická veta $>$ | $<$ zložená veta $>$
$<$ atomická veta $>$	::=	$\top$ | $\bot$ | $<$ symbol $>$
$<$ zložená veta $>$	::=	$<$ unárny operátor $>$ $<$ veta $>$
	|	$<$ veta $>$ $<$ binárny operátor $>$ $<$ veta $>$
	|	($<$ veta $>$)
$<$ unárny operátor $>$	::=	$\neg$
$<$ binárny operátor $>$	::=	$\vee$ | $\wedge$ | $\rightarrow$ | $\leftrightarrow$

Tab. 1.1: Syntax výrokového počtu

Pre vyjadrenie pravdivostnej hodnoty sa používajú dva špeciálne symboly, reprezentujúce logickú pravdu ($\top$) a logickú nepravdu ($\bot$). Okrem týchto dvoch znakov medzi základné stavebné prvky patria ešte symboly, syntax ktorých sa môže rôzniť. Najčastejšie sa označujú pomocou veľkých písmen (napr. P, Q, R, ...) prípadne v tvare písmena doplneného jedným alebo viacerými indexmi, pozostávajúcimi z jednej alebo niekoľkých číslic (napr. P_1, V_{325}, $X_{21,32}$). Indexový tvar sa používa hlavne v prípade väčšieho počtu symbolov, keď počet dostupných samostatných písmen už nepostačuje alebo kvôli názornejšej väzbe na riešený problém. SYMBOLY A ICH OZNAČOVANIE

Unárne operátory majú modifikačnú funkciu, pretože menia význam viet na ktoré sú aplikované. Aj keď existuje viac možných unárnych operá- LOGICKÉ OPERÁTORY

ILUSTRAČNÝ PRÍKLAD

Úloha: *Je potrebné určiť, ktorý zo študentov pri skúške podvádzal. Podozrivými sú Fero, Ondro, Stano, Rudo a Jaro. Tu sú ich výpovede:*

1. *Stano: podvádzali Fero alebo Ondro, avšak nie obaja*
2. *Fero: podvádzali Rudo alebo Stano, avšak nie obaja*
3. *Ondro: podvádzali Jaro alebo Fero, avšak nie obaja*
4. *Rudo: podvádzali Ondro alebo Jaro, avšak nie obaja*
5. *Jaro: podvádzali Ondro alebo Rudo, avšak nie obaja*

V skutočnosti však iba štyri z týchto piatich výpovedí bolo pravdivých, jedna bola nepravdivá. Navyše je k dispozícii ešte výpoveď dozorujúceho učiteľa, ktorú považujme za pravdivú:

6. *Učiteľ: ak Rudo podvádzal, tak potom podvádzal aj Ondro*

Reprezentácia: *Uvedený problém je reprezentovaný ako zložená logická veta tvaru:*

$$
\begin{aligned}
& (\neg P_1 \vee F \vee O) \wedge (\neg P_1 \vee \neg F \vee \neg O) \wedge (\neg F \vee O \vee P_1) \wedge (F \vee \neg O \vee P_1) \\
\wedge\ & (\neg P_2 \vee R \vee S) \wedge (\neg P_2 \vee \neg R \vee \neg S) \wedge (\neg R \vee S \vee P_2) \wedge (R \vee \neg S \vee P_2) \\
\wedge\ & (\neg P_3 \vee J \vee F) \wedge (\neg P_3 \vee \neg J \vee \neg F) \wedge (\neg J \vee F \vee P_3) \wedge (J \vee \neg F \vee P_3) \\
\wedge\ & (\neg P_4 \vee O \vee J) \wedge (\neg P_4 \vee \neg O \vee \neg J) \wedge (\neg O \vee J \vee P_4) \wedge (O \vee \neg J \vee P_4) \\
\wedge\ & (\neg P_5 \vee O \vee R) \wedge (\neg P_5 \vee \neg O \vee \neg R) \wedge (\neg O \vee R \vee P_5) \wedge (O \vee \neg R \vee P_5) \\
\wedge\ & (\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee \neg P_4 \vee \neg P_5) \\
\wedge\ & (P_1 \vee P_2) \wedge (P_1 \vee P_3) \wedge (P_1 \vee P_4) \wedge (P_1 \vee P_5) \wedge (P_2 \vee P_3) \\
\wedge\ & (P_2 \vee P_4) \wedge (P_2 \vee P_5) \wedge (P_3 \vee P_4) \wedge (P_3 \vee P_5) \wedge (P_4 \vee P_5) \\
\wedge\ & (\neg R \vee O)
\end{aligned}
$$

Riešenie: *Do úvahy prichádzajú tri riešenia, keď podvádzajúcimi študentmi boli Ondro a Stano, a to buď sami alebo v spolupáchateľstve s Ferom alebo Jarom.*

Pr. 1.1: Príklad pre reprezentáciu pomocou výrokovej logiky

torov, najčastejšie sa používa operátor *negácie* ($\neg$). Budovať komplexnejšie výrazy umožňujú binárne operátory, ktoré poskytujú možnosť kombinovať prvky logických viet navzájom do zložitejších viet. Aj v prípade týchto binárnych operátorov sa typicky používa iba obmedzená množina operátorov. Štyri najčastejšie používané operátory sú *konjunkcia* ($\vee$), *disjunkcia* ($\wedge$), *implikácia* ($\rightarrow$) a *ekvivalencia* ($\leftrightarrow$).

PRECEDENCIA

Zátvorky slúžia na explicitné vyjadrenie štruktúry logických viet. Ak nie sú použité, tak štruktúra je daná pomocou *precedencie* (priority) ope-

rátorov. Keďže operátory je možné podľa precedencie zoradiť v zmysle od najprioritnejšieho po najmenej prioritný do postupnosti $\neg$, $\wedge$, $\vee$, $\rightarrow$ a $\leftrightarrow$, tak potom veta

$$P \wedge \neg Q \vee R$$

bude reprezentovať vetu $(P \wedge (\neg Q)) \vee R$ a nie napríklad vetu $P \wedge (\neg(Q \vee R))$ – v prípade, keď je potrebné vyjadriť túto druhú štruktúru, je potrebné explicitne použiť zátvorky.

PRAVDIVOSTNÉ HODNOTY

Výroková logika pozná iba dve pravdivostné hodnoty – hodnotu *TRUE* (logická pravda) a hodnotu *FALSE* (logická nepravda). Sémantika definuje vzťah medzi logickými vetami a týmito pravdivostnými hodnotami – sémantika nejakej vety výrokovej logiky definuje, aká pravdivostná hodnota má byť priradená tejto vete v nejakej interpretácii, teda či táto veta je pravdivá (nadobúda hodnotu TRUE) alebo nepravdivá (nadobúda hodnotu FALSE) v danej interpretácii.

Pre ľubovoľnú vetu F a ľubovoľnú interpretáciu I, pravdivostná hodnota F^I, ktorá je priradená vete F interpretáciou I sa určí rekurzívnym spôsobom:

INTERPRETAČNÉ PRAVIDLÁ

- ak F je symbol, potom $F^I = I(F)$
- ak $F = \bot$, tak $F^I = \text{FALSE}$
- ak $F = \top$, tak $F^I = \text{TRUE}$
- ak $F = (\neg G)$, tak $F^I = \neg(G^I)$
- ak $F = (G \odot H)$ a $\odot$ reprezentuje jeden z definovaných binárnych operátorov, tak $F^I = (G^I) \odot (H^I)$.

Teda špeciálny symbol $\top$ je pravdivý v každej interpretácii a $\bot$ je zase v každej interpretácii nepravdivý. Pravdivosť zloženej vety vytvorenej niektorým z operátorov je daná pravdivosťou zložiek tejto vety a predpisom, ako sa z pravdivostných hodnôt zložiek odvádza výsledná pravdivostná hodnota. Takýto predpis je definovaný osobitne pre každý operátor vo forme *pravdivostnej tabuľky*[1] podľa Tab. 1.2. Pravdivostná hodnota symbolu závisí od použitej interpretácie.

PRAVDIVOSTNÉ TABUĽKY

Ak by sme teda napríklad chceli zistiť pravdivostnú hodnotu vety tvaru $P \wedge (\neg(Q \vee R))$ zloženej z troch symbolov a troch operátorov, tak by sme

[1] Keďže existujú dve pravdivostné hodnoty, je možné definovať $2^2 = 4$ rôzne unárne operátory a $4^2 = 16$ rôznych binárnych operátorov.

X	Y	$\neg$ X	X $\wedge$ Y	X $\vee$ Y	X $\rightarrow$ Y	X $\leftrightarrow$ Y
FALSE	FALSE	TRUE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	TRUE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE
TRUE	TRUE	FALSE	TRUE	TRUE	TRUE	TRUE

Tab. 1.2: Pravdivostné tabuľky pre logické operátory

postupne získali

$$\begin{aligned}(P \wedge (\neg(Q \vee R)))^I &= P^I \wedge (\neg(Q \vee R))^I \\ &= P^I \wedge (\neg(Q \vee R)^I) \\ &= P^I \wedge (\neg(Q^I \vee R^I))\end{aligned}$$

a napríklad pri interpretácii $I = \{P^I$=TRUE, Q^I=FALSE, R^I=TRUE$\}$ by po dosadení bolo:

$$\begin{aligned}TRUE \wedge (\neg(FALSE \vee TRUE)) &= TRUE \wedge (\neg TRUE) \\ &= TRUE \wedge FALSE \\ &= FALSE\end{aligned}$$

zatiaľ čo interpretácia $I = \{P^I$=TRUE, Q^I=FALSE, R^I=FALSE$\}$ by viedla na opačnú pravdivostnú hodnotu:

$$\begin{aligned}TRUE \wedge (\neg(FALSE \vee FALSE)) &= TRUE \wedge (\neg FALSE) \\ &= TRUE \wedge TRUE \\ &= TRUE\end{aligned}$$

MOŽNÉ SVETY A MODELY

Jednotlivé interpretácie sa líšia pravdivostnými hodnotami symbolov, pričom každá z nich definuje pravdivostné hodnoty pre všetky použité symboly – interpretácia tak definuje *možný svet*, ktorý je reprezentovaný jednou kombináciou pravdivostných hodnôt použitých symbolov. Keďže každý symbol môže mať iba jednu z dvoch možných hodnôt, tak ak je použitých n rôznych symbolov, potom existuje 2^n rôznych interpretácií. Tá interpretácia, v ktorej je nejaká veta pravdivá, sa označuje ako *model* danej vety.

Ak pre nejakú vetu existuje model, potom táto veta sa označuje ako *splniteľná*. Ak pre ňu žiadny model neexistuje (teda nie je pravdivá pri žiadnej interpretácii), tak sa označuje ako *nesplniteľná*. Veta, ktorá je pravdivá pri ľubovoľnej interpretácii a teda každá možná interpretácia je jej modelom,

sa označuje ako *validná* a nazýva sa *tautológia*. Príkladom tautológie je veta $A \to (B \to (A \to B))$, čo možno overiť pomocou pravdivostnej tabuľky pre túto vetu.

EKVIVALENTNOSŤ VIET

V prípade, že dve vety pre každú možnú interpretáciu nadobúdajú rovnaké pravdivostné hodnoty (teda im prislúchajú rovnaké pravdivostné tabuľky), tak potom tieto vety sú navzájom *ekvivalentné*. Ako znak ekvivalentnosti viet sa používa $\equiv$. Pomocou pravdivostných tabuliek je možné overiť napríklad komutatívnosť konjunkcie $P \wedge Q \equiv Q \wedge P$ či disjunkcie $P \vee Q \equiv Q \vee P$ alebo na druhej strane ukázať, že implikácia nie je komutatívna. Podobne je možné overiť napríklad asociatívnosť konjunkcie $(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$ ako aj disjunkcie $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$ alebo platnosť pohlcovacích (absorpčných) pravidiel ($P \vee P \equiv P$, $P \wedge P \equiv P$, $P \vee \bot \equiv P$, $P \wedge \top \equiv P$, $P \wedge (P \vee Q) \equiv P$, $P \vee (P \wedge Q) \equiv P$, $P \vee \top \equiv \top$, $P \wedge \bot \equiv \bot$, $P \vee (\neg P \wedge Q) \equiv P \vee Q$, $P \wedge (\neg P \vee Q) \equiv P \wedge Q$). Vďaka ekvivalentnosti viet je možné jeden operátor nahradiť pomocou iných operátorov. V skutočnosti pomocou dvojice $\neg$ a $\vee$ (alebo dvojice $\neg$ a $\wedge$) je možné vyjadriť každý zo šestnástich možných binárnych operátorov.

UKOTVENIE V REALITE

Použitie logiky pre reprezentáciu znalostí nezaručuje automaticky, že získané výsledky budú správne. Výroková logika predstavuje formálny systém, garantujúci validitu záverov – ale iba v prípade validnosti predpokladov, z ktorých sa vychádza. Pre praktické použitie je preto dôležité, akým spôsobom budú symboly ako aj zložené vety *ukotvené* v realite (reprezentovanej doméne). Reprezentácia faktov a vzťahov, platiacich v nejakej doméne, do tvaru logických viet nie je nijako špeciálne náročná, treba však mať na mysli rozdiely medzi významom týchto faktov a vzťahov (najmä ak nie sú vyjadrené prísne formálnym spôsobom) a sémantikou reprezentačných nástrojov, ponúkaných výrokovou logikou. Ukážme si ukotvenie na ilustračnom prípade Pr. 1.1.

GRANULARITA

Voľba symbolov môže byť rôzna, avšak mala by sa riadiť účelom reprezentácie a voľbou vhodnej granularity. Je síce možné symbolom reprezentovať ľubovoľnú časť domény, avšak nie vždy to je najvhodnejšia voľba. Niekoľko príkladov ukotvenia symbolov:

> "Fero podvádza" – táto voľba umožňuje uvažovať o Ferovom podvádzaní samostatne a prípadne ho pomocou zložitejších viet spájať s inými skutočnosťami.
>
> "Fero alebo Ondro podvádzajú" – neumožňuje pracovať s Ferovým podvádzaním bez uvažovania Ondrovho podvádzania, nedá sa vyjadriť súvislosť so symbolom reprezentujúcim "Jaro alebo Fero podvádzajú", hoci tieto dva symboly sú navzájom závislé cez Ferovo podvádzanie.

Rozšírenie: Reprezentácia premenných

KÓDOVANIE PREMENNÝCH

Pri reprezentovaní úloh sa možno často stretnúť s prípadom, keď v popise problému vystupuje nejaká premenná, ktorej doména je tvorená množinou vymenovaných hodnôt, ktoré premenná môže nadobúdať. Ak je takáto premenná binárna (môže nadobúdať iba jednu z dvoch možných hodnôt), tak takúto premennú je možné reprezentovať priamo symbolom – ak symbol platí, tak premenná nadobudla jednu z hodnôt, a ak symbol neplatí, tak premenná nadobudla druhú hodnotu.

Kódovací problém nastáva, ak doména je tvorená $n > 2$ hodnotami. Pri *priamom* kódovaní sa použije n symbolov, pričom každý reprezentuje jednu z hodnôt – symbol X_i bude reprezentovať skutočnosť, že premenná nadobudla i-tu hodnotu H_i zo svojej domény. Ku kódovaniu je potom ešte potrebné pridať podmienku, že premenná musí nadobudnúť práve jednu hodnotu.

Pri *logaritmickom kódovaní* je potrebných pre jednu premennú $m = \lceil log_2 n \rceil$ symbolov. Vzťah medzi symbolmi, použitými pre reprezentáciu hodnôt domény, a samotnými hodnotami je zložitejší. Pre zjednodušenie uvažujme konkrétny príklad domény s piatimi hodnotami H_1 až H_5. Pre ich kódovanie sú potrebné tri symboly X_1 až X_3 podľa nasledujúcej tabuľky:

	X_1	X_2	X_3
H_1	FALSE	FALSE	FALSE
H_2	FALSE	FALSE	TRUE
H_3	FALSE	TRUE	FALSE
H_4	FALSE	TRUE	TRUE
H_5	TRUE	FALSE	FALSE

	X_1	X_2	X_3
H_1	FALSE	FALSE	FALSE
H_2	FALSE	FALSE	TRUE
H_3	FALSE	TRUE	
H_4	TRUE	FALSE	
H_5	TRUE	TRUE	

Fakt, že premenná nadobudla nejakú hodnotu, sa vyjadrí ako konjunkcia. Buď sa použije vždy všetkých m symbolov (neoptimalizovaná podoba – ľavá tabuľka) alebo pre niektoré hodnoty sa použije m a pre iné zase $m - 1$ symbolov (optimalizovaná podoba – pravá tabuľka). Teda napríklad použitie hodnoty H_4 sa vyjadrí buď ako $\neg X_1 \wedge X_2 \wedge X_3$ alebo ako $X_1 \wedge \neg X_2$. Výhodou logaritmického kódovania je menší počet potrebných symbolov. Pri neoptimalizovanej podobe je nutné ešte kódovať podmienku, že premenná musí nadobudnúť aspoň jednu hodnotu (teda že napríklad $X_1 \wedge X_2 \wedge X_3$ nie je prípustné). To, že nesmie nadobudnúť súčasne dve hodnoty, je už zabezpečené samotným kódovaním. Pri optimalizovanej podobe už nie je potrebné kódovať žiadnu dodatočnú podmienku.

> "Stanovo tvrdenie" – vhodné v prípade, že je potrebné uvažovať o Stanovom tvrdení ako celku, teda o tom aká je jeho pravdivosť (reprezentovať "Učiteľovo tvrdenie" je možné avšak zbytočné, keďže jeho pravdivosť je známa).

Vhodnou voľbou v danom prípade by teda bolo reprezentovať podvádzania osobitne (napr. symbol F pre "Fero podvádza", O pre "Ondro podvádza", atď.) a aj výroky ako celky okrem šiesteho výroku (napr. symbol P_1 pre prvý výrok, P_2 pre druhý výrok). Hoci symboly môžu byť označené ľubovoľne, je vhodné použiť označenia evokujúce čo daný symbol označuje.

Kódovanie "A", "Aj"

Tvrdenia, využívajúce spojky typu "a" či "aj" s významom existencie viacerých konceptov, ktoré sú súčasne pravdivé, je možné reprezentovať pomocou $\wedge$ operátora. Napríklad súčasná pravdivosť Stanovho a Ferovho výroku by mohla byť reprezentovaná ako $P_1 \wedge P_2$, kde symbol P_1 reprezentuje Stanov výrok a symbol P_2 zase Ferov výrok.

Kódovanie "Alebo"

Zložitejšie je to v prípade spojky "alebo" – tejto spojke môžu zodpovedať dva operátory. Jedným je $\vee$, ktorý sa niekedy označuje ako *inclusive-or* so zdôraznením, že zahŕňa nielen platnosť každého z argumentov ale aj ich súčasnú platnosť. Na rozdiel od neho operátor označovaný ako *exclusive-or* zahŕňa platnosť každého z argumentov samostatne ale vylučuje platnosť oboch argumentov súčasne. To, ktorý z nich má byť použitý, závisí od toho ako má byť interpretovaný prípad súčasnej platnosti argumentov. Keďže Stanov výrok P_1 explicitne vylučuje, aby Fero a Ondro podvádzali súčasne, tak toto tvrdenie je možné reprezentovať ako

$$F \; xor \; O \equiv (F \vee O) \wedge \neg(F \wedge O)$$

s významom, že platí že Fero alebo Ondro podvádza a zároveň neplatí, že podvádzajú obaja súčasne.

Kódovanie "Ak-potom"

Podobne je potrebné byť obozretným aj v prípade viet tvaru "ak ... potom ..." vyjadrujúcich, že nejaký predpoklad má za následok nejaký záver. Aj v tomto prípade sú možné dve rôzne chápania zmyslu tvrdenia. Prvé za pravdivé pokladá nielen explicitne vyjadrenú závislosť na platnosti predpokladu ale aj implicitne vyjadrenú závislosť na neplatnosti predpokladu (v prípade ktorej záver neplatí) – to je chápanie v zmysle "vtedy a iba vtedy záver ak predpoklad". V tomto prípade je vhodnou voľbou $\leftrightarrow$ operátor. Druhé chápanie je založené iba na uvažovaní explicitne vyjadrenej závislosti bez nejakého implicitného dodatku pod povrchom. V takomto prípade je voľbou $\rightarrow$ operátor. Ak budeme učiteľov výrok chápať iba v explicitne podanom zmysle, potom šieste tvrdenie bude reprezentované ako

$$R \rightarrow O$$

kde symbol R reprezentuje Rudovo podvádzanie a symbol O analogicky Ondrovo podvádzanie.

KÓDOVANIE PLATNOSTI VÝROKOV

Zložitejšia situácia nastáva v prípade výrokov o iných výrokoch. V našom prípade veta F *xor* O zo Stanovho výroku P_1 platí iba vtedy, ak Stanov výrok je pravdivý. Ak je však jeho výrok nepravdivý, tak táto veta nie je pravdivá. Na druhej strane, ak je pravdivý obsah Stanovho výroku P_1, tak je pravdivý aj samotný výrok. Preto Stanov výrok je potrebné reprezentovať ako

$$P_1 \leftrightarrow F \; xor \; O \equiv (P_1 \rightarrow (F \vee O) \wedge \neg(F \wedge O)) \wedge ((F \vee O) \wedge \neg(F \wedge O) \rightarrow P_1)$$

KÓDOVANIE "ASPOŇ" / "NAJVIAC JEDEN"

Požiadavka, aby jedno z piatich tvrdení bolo nepravdivé, sa dá vyjadriť ako dvojica *kardinalitných ohraničení* – aspoň jedno tvrdenie je nepravdivé a súčasne najviac jedno z tvrdení je nepravdivé. Vyjadriť prvé je možno jednoducho pomocou

$$\neg P_1 \vee \neg P_2 \vee \neg P_3 \vee \neg P_4 \vee \neg P_5$$

zatiaľ čo vyjadrenie druhého ohraničenia je zložitejšie. Je možné vyjsť z toho, že žiadne dve z tvrdení nesmú byť súčasne nepravdivé (a teda v rámci každej možnej dvojice tvrdení musí byť aspoň jedno tvrdenie pravdivé).

KONJUNKTÍVNA NORMÁLNA FORMA

Vďaka ekvivalentnosti logických výrazov je možné logické vety transformovať z jedného tvaru na iný tvar. Jeden z možných tvarov, ktorý sa široko používa, sa označuje ako CNF (*konjunktívna normálna forma*). Definícia tejto formy je v Tab. 1.3. Charakteristickými vlastnosťami tohto tvaru sú:

- používajú sa iba dva binárne operátory a to $\wedge$ a $\vee$, iné binárne operátory nie sú povolené,
- negované môžu byť iba symboly, zložené vety nie sú negované,
- disjunkcia je prípustná v rámci konjunkcie avšak konjunkcia v rámci disjunkcie prípustná nie je.

$<$ veta $>$	::=	$<$ klauzula $>$
	$\mid$	$<$ klauzula $> \wedge <$ veta $>$
$<$ klauzula $>$	::=	$<$ literál $>$
	$\mid$	$<$ literál $> \vee <$ klauzula $>$
$<$ literál $>$	::=	$\neg <$ symbol $> \mid <$ symbol $>$

Tab. 1.3: Konjunktívna normálna forma

Rozšírenie: Kardinalitné ohraničenia

Kódovanie kardinalitných ohraničení

Často je potrebné vyjadriť obmedzenie na platnosť iba určitej časti symbolov z nejakej množiny symbolov. Formálne to je možné vyjadriť

$\leq_k (X_1, X_2, \ldots, X_n)$ – maximálne k symbolov z n môže byť pravdivých a teda ostatné z danej množiny musia byť nepravdivé

$\geq_k (X_1, X_2, \ldots, X_n)$ – minimálne k symbolov z n musí byť pravdivých

$=_k (X_1, X_2, \ldots, X_n)$ – práve k symbolov z n je pravdivých.

Keďže podmienka "práve k" môže byť vyjadrená ako súčasná platnosť ostatných dvoch podmienok a navyše keďže "aspoň" a "najviac" sú ekvivalentné podľa

$$\geq_k (X_1, X_2, \ldots, X_n) \quad = \quad \leq_{n-k} (\neg X_1, \neg X_2, \ldots, \neg X_n)$$

tak je postačujúce sa zaoberať iba "najviac k" podmienkou. Táto platí pre $0 < k < n$ ($k = 0$ môže byť jednoducho reprezentované pomocou n jednotkových klauzúl, $k = n$ reprezentuje vždy splnené ohraničenie, ktoré možno vynechať). Túto podmienku možno reprezentovať viacerými spôsobmi, z ktorých je najjednoduchším *binomické kódovanie*.

Zo sémantiky podmienky vyplýva, že pre nejakú hodnotu k nie je povolené, aby všetky symboly $X_1, \ldots, X_k, X_{k+1}$ boli pravdivé – aspoň jeden z nich musí byť nepravdivý. Toto môže byť reprezentované ako

$$\neg X_1 \vee \ldots \vee \neg X_k \vee \neg X_{k+1}$$

čo vylučuje porušenie podmienky. A pretože toto platí pre každú možnú kombináciu $k+1$ symbolov, je potrebné generovať takúto klauzulu pre všetky kombinácie $k+1$ symbolov

$$\bigwedge_{M \subseteq \{1,\ldots,n\}, |M|=k+1} \bigvee_{i \in M} \neg X_i$$

čo vyžaduje $\begin{pmatrix} n \\ k+1 \end{pmatrix}$ klauzúl dĺžky $k+1$.

Nevýhodou tohto kódovania je veľký počet generovaných klauzúl, preto existuje mnoho ďalších kódovaní (napr. *binárne*, *sekvenčné*, *commander*, *product*, *bimander*, *kardinalitné siete*, atď.), ktoré znižujú počet klauzúl za cenu definovania dodatočných pomocných symbolov.

Príkladmi syntakticky správnych viet v tvare CNF sú P, $\neg P$, $P \vee Q$, $P \wedge Q$ či $(\neg P \vee Q) \wedge \neg R$. Validnými sú všetky vety pozostávajúce z konjunkcie klauzúl, ktoré sú reprezentované ako disjunkcie literálov. Samostatná konjunkcia je v CNF (klauzuly majú jednotkovú dĺžku) rovnako ako samostatná disjunkcia (iba jedna klauzula) alebo samostatný symbol či už negovaný alebo bez negácie (iba jedna klauzula jednotkovej dĺžky).

PREVOD DO CNF

Ľubovoľná veta vo výrokovom počte môže byť transformovaná na ekvivalentnú vetu v tvare CNF. Neznamená to však vždy zjednodušenie v zmysle počtu prvkov zápisu, výsledkom môže byť aj nárast zložitosti. Príkladom tohto je veta

$$(X_1 \wedge Y_1) \vee (X_2 \wedge Y_2)$$

ktorej zodpovedá ekvivalentná CNF tvaru

$$(X_1 \vee X_2) \wedge (X_1 \vee Y_2) \wedge (Y_1 \vee X_2) \wedge (Y_1 \vee Y_2)$$

vstup: veta F v ľubovoľnom tvare
výstup: veta F v CNF tvare

1. $F :=$ *eliminácia_ekvivalencií(F)*
2. $F :=$ *eliminácia_implikácií(F)*
3. $F :=$ *vnorenie_negácií(F)*
4. $F :=$ *úprava_na_CNF(F)*

Alg. 1.1: Transformácia do CNF

ALGORITMUS PRE PREVOD DO CNF

Algoritmus pre transformáciu logického výrazu do CNF pozostáva zo štyroch krokov (Alg. 1.1), pričom každý z nich je založený na využití jedného alebo viacerých *odvodzovacích pravidiel*. Ekvivalencie sú eliminované na základe

$$P \leftrightarrow Q \;\equiv\; (P \rightarrow Q) \wedge (Q \rightarrow P)$$

zatiaľ čo implikácie je možné odstrániť využitím ekvivalentnosti

$$P \rightarrow Q \;\equiv\; \neg P \vee Q$$

Pre realizáciu tretieho kroku je potrebné využiť De Morganove pravidlá

$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q \qquad \neg(P \vee Q) \equiv \neg P \wedge \neg Q$$

a pravidlo dvojitej negácie

$$\neg(\neg P) \equiv P$$

Výsledná úprava na CNF tvar je založená na distributívnosti konjunkcie nad disjunkciou respektíve disjunkcie nad konjunkciou

$$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R) \qquad P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$$

ako aj na komutatívnosti a asociatívnosti konjunkcie a disjunkcie.

Ako ukážku postupnej úpravy je možné upraviť prepis Stanovho výroku z ilustračného príkladu Pr. 1.1, ktorého prvá časť má tvar

$$P_1 \rightarrow (F \vee O) \wedge \neg(F \wedge O)$$

Po nahradení implikácie sa získa $\neg P_1 \vee ((F \vee O) \wedge \neg(F \wedge O))$, prechod na tvar $\neg P_1 \vee ((F \vee O) \wedge (\neg F \vee \neg O))$ bol dosiahnutý aplikáciou jedného z De Morganových pravidiel a $(\neg P_1 \vee (F \vee O)) \wedge (\neg P_1 \vee (\neg F \vee \neg O))$ je výsledkom využitia distributívnosti disjunkcie nad konjunkciou. Následným využitím asociatívnosti sa konečne získa $(\neg P_1 \vee F \vee O) \wedge (\neg P_1 \vee \neg F \vee \neg O)$.

Pri použití výrokovej logiky pre reprezentáciu znalostí je možné vyčleniť dva hlavné typy úloh, ktoré je potrebné riešiť: HLAVNÉ TYPY ÚLOH

- dedukčná úloha,
- úloha splniteľnosti.

Pri *dedukčnej úlohe* sú znalosti o probléme (znalostná báza KB reprezentujúca súhrn dostupných znalostí reprezentovaný ako konjunkcia týchto znalostí) reprezentované vetou vo výrokovom počte. Cieľom je nájsť ďalšie znalosti, ktoré zo známych znalostí vyplývajú. Predpokladá sa, že báza znalostí KB je pravdivá (teda implicitne sa uvažujú iba tie interpretácie, v ktorých nadobúda hodnotu logickej pravdy) a hľadá sa nejaká znalosť F taká, pre ktorú bude platiť $KB \models F$, kde znak $\models$ reprezentuje *logické vyplývanie*. V ilustračnom príklade Pr. 1.1 by takýmito novými znalosťami mohli byť napríklad: "Fero hovorí pravdu", "Stano podvádzal", "ak Fero hovorí pravdu potom aj Jaro hovorí pravdu" alebo "podvádzali Stano alebo Rudo, avšak nie obaja" či "ak Rudo podvádzal, potom ak Fero podvádzal tak aj Ondro podvádzal". DEDUKČNÁ ÚLOHA

LOGICKÉ VYPLÝVANIE

Formálne logické vyplývanie $P \models Q$ (Q vyplýva z P) je definované tak, že vo všetkých tých svetoch, v ktorých je P pravdivé, je pravdivé aj Q – teda každý model vety P je súčasne aj modelom vety Q. Opačne to však nemusí platiť, model vety Q nemusí byť modelom vety P. Príkladom logického vyplývania je odvodenie tvaru

REZOLVENCIA

$$\frac{P \vee Q \qquad \neg Q \vee R}{P \vee R}$$

reprezentujúce *rezolvenčné odvodzovacie pravidlo*, podľa ktorého z platnosti dvoch disjunkcií, obsahujúcich komplementárne vyjadrenie literálu nejakej premennej (prítomnej ako v priamej tak aj v negovanej podobe) je možné odvodiť platnosť disjunkcie vzniknutej spojením oboch vstupných disjunkcií s vynechaním oboch komplementárnych literálov predmetnej premennej. Špecifickým prípadom rezolvencie je situácia, keď jedným zo vstupných disjunkcií je samostatný literál

$$\frac{P \qquad P \rightarrow R}{R}$$

pričom tento tvar je známy ako *modus ponens*.

Dedukcia má charakter overovania – nová znalosť reprezentuje hypotézu, ktorú je potrebné dokázať. Dokazovanie môže byť priame alebo sporom, pričom sa využívajú rôzne inferenčné pravidlá, napríklad rôzne typy eliminácií, modus ponens alebo dedukčný teorém. Týmto typom úlohy sa tu nebudeme zaoberať, pretože sa mu podrobnejšie venuje kapitola 2 venovaná predikátovej logike[2].

SAT ÚLOHA

V prípade *úlohy splniteľnosti*, ktorá je známa aj pod označením SAT (SATisfiability) problém, sú znalosti o riešenom probléme opäť vyjadrené vo forme vety vo výrokovom počte. Cieľom je nájsť takú interpretáciu, pri ktorej daná veta reprezentujúca znalosti nadobúda hodnotu logickej pravdy. Každá takáto interpretácia, charakterizovaná nejakým priradením logických hodnôt symbolom, je riešením úlohy. Alternatívne sa niekedy za cieľ považuje namiesto jednej vhodnej interpretácie nájsť všetky vhodné interpretácie alebo namiesto ich nájdenia určiť ich počet.

Jedným z existujúcich prístupov ako riešiť takúto úlohu je systematické prehľadávanie priestoru všetkých možných interpretácií. Tento prístup je založený na postupnom prehľadávaní s navracaním na základe stratégie *prehľadávania do hĺbky*. Zosobnením tohto prístupu je *DPLL algoritmus*

[2]Tam popísaný postup dokazovania vhodný pre predikátovú logiku môže byť v zjednodušenej podobe použitý aj pri výrokovej logike.

vstup: veta F v CNF tvare
výstup: interpretácia I v prípade úspechu alebo {} pri neúspechu

```
1.  s := 0, I := {}
2.  loop
3.      switch F^I
4.          case TRUE
5.              return I
6.          case FALSE
7.              while s > 0  and flip[s] := 1
8.                  s := s − 1
9.              I := I \ {I(X)_t : t > s}
10.             if s = 0 then return {}
11.                      else flip[s] := 1
12.                           I(X)_s := ¬I(X)_s
13.         default
14.             s := s + 1
15.             X := select_symbol(F,I)
16.             flip[s] := 0
17.             I := I ∪ I(X)_s
```

Alg. 1.2: DPLL skeleton

(spojený s menami Davis, Putnam, Logemann a Loveland), ktorý sa nachádza v jadre mnohých dnešných *SAT solverov*. Základná schéma algoritmu je uvedená v Alg. 1.2.

DPLL algoritmus

Jedná sa o postupné budovanie parciálnej (neúplnej) interpretácie. Začína sa z prázdnej interpretácie $I = \{\}$, do ktorej sa postupne v nasledujúcich krokoch pridávajú interpretácie symbolov $I(X)_s$, kde X reprezentuje interpretovaný symbol, $I(X)$ jeho interpretáciu a s reprezentuje číslo kroku, v ktorom bola daná interpretácia pridaná. Je možné pridať interpretáciu symbolu ako pravdivého alebo nepravdivého – záznam o tom, aká interpretácia bola pridaná, sa udržiava v poli $flip$. Hodnota $flip[s] = 0$ znamená, že v kroku s bola pridaná prvá interpretácia zvoleného symbolu, zatiaľ čo

hodnota $flip[s] = 1$ signalizuje, že pre daný symbol bola jeho interpretácia zmenená na druhú interpretáciu (a teda už boli vyčerpané všetky možnosti pre daný symbol). Takýmto postupom sa parciálna interpretácia rozširuje až nastane jedna zo situácií:

- na základe interpretácie symbolov je možné interpretovať celú vetu F ako pravdivú,
- vetu F na základe interpretácie symbolov je možné interpretovať ako nepravdivú,
- na základe interpretácie symbolov vetu F ešte nie je možné interpretovať ako celok.

Týmto trom situáciám zodpovedajú tri vetvy algoritmu. V prípade prvej situácie (riadok 4) vytvorená interpretácia predstavuje hľadané riešenie a preto prehľadávanie končí. Druhá situácia (riadok 6) znamená, že aktuálnu parciálnu interpretáciu nie je možné rozšíriť tak, aby reprezentovala hľadané riešenie. Je potrebné sa vrátiť spätne po vykonaných krokoch (s odstránením posledne vykonaných interpretácií symbolov) až ku kroku so symbolom, pre ktorý neboli vyskúšané obe možné pravdivostné hodnoty. Ak taká možnosť nie je (pokus o *navracanie* skončí prípadom $s = 0$), signalizuje sa nemožnosť nájdenia riešenia. Ak sa však podarí nájsť nejaký predošlý krok so symbolom, pre ktorý bola vyskúšaná iba jedna z možných interpretácií, tak pre daný symbol sa zmení jeho interpretácia a hľadanie opäť pokračuje od tohto symbolu avšak už so zmenenou interpretáciou. V tretej situácii (riadok 13) dochádza k ďalšiemu rozšíreniu parciálnej interpretácie symbolov. Zvolí sa ďalší symbol (či už ľubovoľne alebo ďalší vo vopred dohodnutom poradí) a jemu priradená pravdivostná hodnota (prvá vo vopred dohodnutom poradí). Voľba sa zaregistruje v poli $flip$ a zaradí do interpretácie.

Skeleton algoritmu v sebe obsahuje dve slučky:

- hlavná slučka (riadky 2 až 17),
- navracacia slučka (riadky 7 až 8).

pričom v rámci hlavnej slučky sa parciálna interpretácia symbolov rozširuje o nové symboly, zatiaľ čo v navracacej slučke sa parciálna interpretácia symbolov skracuje.

Rozšírenia DPLL

Moderné SAT solvery založené na systematickom prehľadávaní tento základný algoritmus dopĺňajú o ďalšie prvky, umožňujúce zrýchlenie nájdenia riešenia. Ako možné doplnenia sa najčastejšie používajú:

- jednotková propagácia a propagácia čistého literálu,
- učenie klauzúl riadené konfliktami,
- nechronologické navracanie,
- heuristiky pre výber symbolu a hodnoty.

JEDNOTKOVÁ PROPAGÁCIA

Jednotková propagácia je založená na využití konceptu jednotkovej klauzuly – klauzuly, ktorá obsahuje iba jeden literál. Aby takáto klauzula bola pravdivá, musí byť pravdivý jej literál a teda je zrejmé, aká pravdivostná hodnota musí byť zvolená pre interpretáciu daného symbolu (v závislosti od toho, či symbol je v literále negovaný alebo nie). Keďže priradenie pravdivostnej hodnoty symbolu môže mať za následok, že nejaká iná klauzula sa stane jednotkovou, tak takáto propagácia môže mať rekurzívny charakter. Príkladom sú klauzuly

$$(\neg P \vee Q) \quad \wedge \quad (\neg P \vee Q \vee \neg R) \quad \wedge \quad P$$

kde tretia klauzula je jednotková a preto $I(P) = \text{TRUE}$. Následkom toho prvý literál v prvej klauzule je nepravdivý (a teda môže byť z klauzuly vypustený) a prvá klauzula sa tak stáva tiež jednotkovou s následkom $I(Q) = \text{TRUE}$. Výsledkom je, že všetky tri klauzuly sú interpretované ako pravdivé (bez špecifikovania interpretácie symbolu R, pretože v splnených klauzulách nie je potrebné skúmať zostávajúce literály symbolov bez interpretácie).

PROPAGÁCIA ČISTÉHO LITERÁLU

Podobný charakter má *propagácia čistého literálu*. O čistom literále sa hovorí vtedy, ak nejaký symbol sa v klauzulách vyskytuje iba v jednej podobe, buď iba priamo ako jednoduchý symbol alebo iba negovane ako negovaný symbol. Vtedy je možné daný symbol interpretovať tak, aby vyskytujúci sa literál zapríčinil pravdivosť tých klauzúl, v ktorých sa nachádza. Takáto interpretácia nemá za následok nepravdivosť žiadneho literálu, pretože druhá podoba literálu daného symbolu sa v klauzulách nenachádza. Príkladom sú klauzuly

$$(\neg P \vee Q) \quad \wedge \quad (\neg P \vee S) \quad \wedge \quad (\neg Q \vee R) \quad \wedge \quad (\neg S \vee \neg R)$$

kde symbol P sa vyskytuje iba v podobe negovaného literálu, zatiaľ čo ostatné symboly vytvárajú literály oboch podôb. Interpretácia $I(P) = \text{FALSE}$ spôsobí, že prvé dve klauzuly sú interpretované ako pravdivé (a teda literály Q a $\neg S$ nemusia byť skúmané). Následkom toho budú $\neg Q$ a $\neg S$ považované za ďalšie čisté literály a teda bude môcť byť použitá interpretácia $I(Q) = \text{FALSE}$ a $I(S) = \text{FALSE}$ so splnením zostávajúcich dvoch klauzúl. Tento typ propagácie môže mať teda tiež rekurzívny charakter.

KOMBINOVANIE PROPAGÁCIÍ

Jednotková propagácia môže vytvoriť podmienky pre propagáciu čistého literálu (naopak to nie je možné). Toto je možné ilustrovať pomocou

$$(\neg P \vee \neg Q \vee \neg S) \quad \wedge \quad (P \vee Q) \quad \wedge \quad P \quad \wedge \quad (S \vee \neg R) \quad \wedge \quad (\neg Q \vee R)$$

kde na základe jednotkovej propagácie bude $P^I = \text{TRUE}$, druhá klauzula bude interpretovaná ako pravdivá a prvá klauzula sa zmení na $(\neg Q \vee \neg S)$. Vďaka neuvažovaniu druhého literálu v druhej klauzule sa symbol Q vyskytuje iba v podobe negovaného literálu a teda je možné na základe propagácie čistého literálu použiť $I(Q) = \text{FALSE}$.

DOPLNENIE DPLL O PROPAGÁCIE

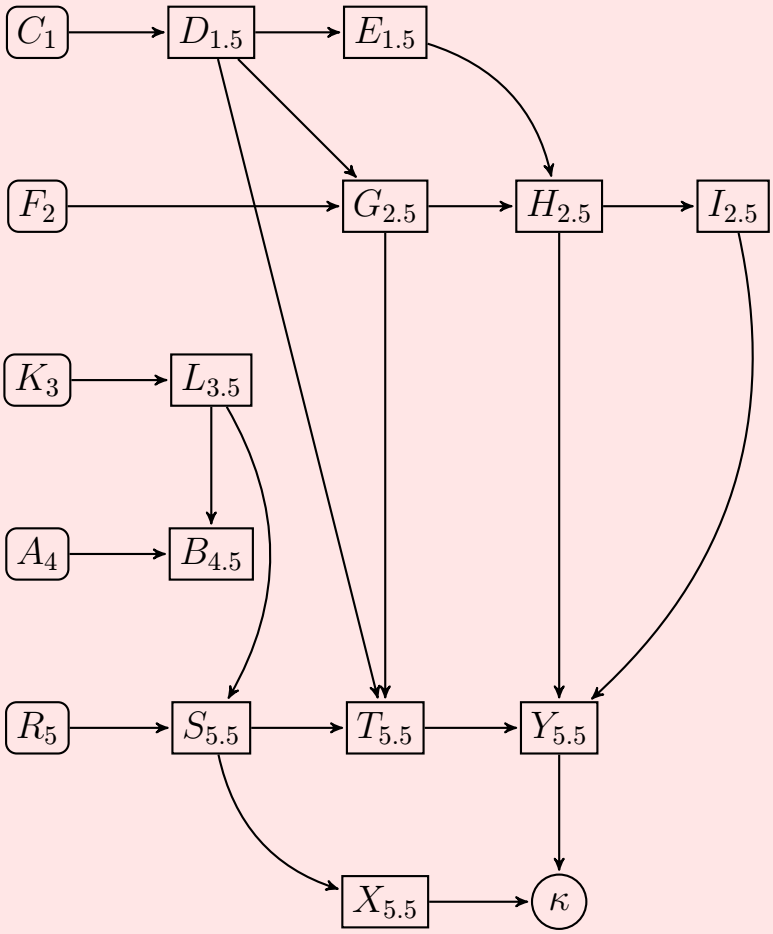
Oba typy propagácie reprezentujú orezávanie alternatív, ktoré je potrebné prehľadať, pretože pre symboly interpretované v rámci týchto propagácií nemá zmysel pri neúspechu skúmať ich alternatívne interpretácie. Je ich možné zaradiť do algoritmu Alg. 1.2 medzi riadky 2 a 3 v tvare

```
2.4.   while I(X)_{s+0.5} := unit_propagation(F)
2.5.       I := I ∪ I(X)_{s+0.5}
2.6.   while I(X)_{s+0.5} := pure_literal_propagation(F)
2.7.       I := I ∪ I(X)_{s+0.5}
```

pričom aby sa využil vzťah medzi obomi typmi propagácie, tak je potrebné najprv zaradiť jednotkovú propagáciu a až po nej propagáciu čistého literálu. Aby bolo možné odlíšiť, interpretácia ktorého symbolu bola v kroku s pridaná do I na základe niektorej z propagácií a ktorá na základe voľby v riadkoch 15 až 17, interpretácie symbolov z propagácií sú pridávané s neceločíselným krokom (aby navracanie v riadkoch 7 a 8, pracujúce iba s celočíselnými hodnotami kroku, sa navracalo iba k tým symbolom, ktorých interpretácie pochádzajú z voľby na riadku 17).

IMPLIKAČNÝ GRAF

Propagácia je prirodzeným spôsobom asociovaná s *implikačným grafom*, ktorý zachytáva všetky spôsoby, ktorými boli určené interpretácie symbolov. Ukážka takéhoto grafu je na Obr. 1.1. Jedná sa o orientovaný acyklický graf, ktorého uzlami sú interpretácie symbolov, pričom každému uzlu zodpovedá práve jeden symbol (interpretácii $I(X)_s = \text{TRUE}$ vytvorenej na úrovni s zodpovedá uzol reprezentujúci literál X_s, zatiaľ čo interpretácii $I(X)_s = \text{FALSE}$ zase uzol reprezentujúci literál $\neg X_s$). Hrany vyjadrujú závislosti medzi interpretáciami, napr. podľa obrázku symbol Y bol interpretovaný na základe interpretácií symbolov H, I a T (možno vďaka klauzule $\neg T \vee \neg H \vee \neg I \vee Y$ alebo možno vďaka klauzulám $\neg T \vee \neg I \vee Y$ a $\neg H \vee Y$). Voliteľne môžu byť tieto hrany označené množinami tých klauzúl, ktoré danú závislosť reprezentujú.



Obr. 1.1: Implikačný graf

Na obrázku sú interpretácie na základe voľby reprezentované obdĺžnikmi s oblými hranami a celočíselnými krokmi, zatiaľ čo interpretácie na základe propagácie sú reprezentované obdĺžnikmi s ostrými hranami a neceločíselnými krokmi. Obrázok teda reprezentuje postup, keď najprv bola zvolená interpretácia symbolu C, následne pomocou propagácie boli určené interpretácie symbolov D a E, nasledovala voľba pre symbol F, za ňou na základe propagácií interpretácie symbolov G, H a I, voľba pre K, atď.

Konfliktný graf

Ak by nejaký symbol mal v tomto grafe nadobudnúť rôzne interpretácie, znamená to konflikt a namiesto týchto interpretácií je v grafe vložený konfliktný uzol κ (konflikt na obrázku vznikol kvôli klauzulám $Z \vee \neg X$

a $\neg Z \vee \neg Y$). Implikačný graf môže obsahovať žiadny, jeden alebo aj viac konfliktných uzlov. V prípade konfliktu je z takéhoto grafu možné odvodiť *konfliktný graf* ako podgraf implikačného grafu, ktorý obsahuje iba jeden konfliktný uzol a súčasne obsahuje iba tie uzly, z ktorých je daný konfliktný uzol dosiahnuteľný. Ak implikačný graf obsahuje viacero konfliktných uzlov, tak potom je možné z neho odvodiť viac navzájom rôznych konfliktných grafov. Na obrázku nie je implikačný graf totožný s konfliktným grafom (uzly A_4 a $B_{4.5}$ sú súčasťou implikačného grafu ale nie konfliktného).

ANALÝZA KONFLIKTU

Konfliktný graf sa používa pre analýzu konfliktu a vyhľadávanie konfliktných klauzúl. Keďže podľa obrázku bezprostrednou príčinou konfliktu bola interpretácia symbolov Y a X (oba boli interpretované ako pravdivé), tak pre zabránenie konfliktu musí platiť klauzula

$$\neg(X \wedge Y) \equiv \neg X \vee \neg Y$$

Z grafu je zrejmé, že interpretácia symbolu Y sa dá získať pomocou implikácie

$$T \wedge H \wedge I \rightarrow Y \equiv \neg T \vee \neg H \vee \neg I \vee Y$$

Z posledne uvedených dvoch klauzúl je možné získať pomocou rezolvenčného odvodzovacieho pravidla (zohľadnením výskytu oboch literálov symbolu Y) ďalšiu klauzulu, ktorá musí platiť aby nedošlo ku konfliktu

$$\frac{\neg X \vee \neg Y \qquad \neg T \vee \neg H \vee \neg I \vee Y}{\neg X \vee \neg T \vee \neg H \vee \neg I}$$

REZY KONFLIKTNÝM GRAFOM

Obe odvodené klauzuly reprezentujú *rezy* konfliktným grafom, keď ho delia na dva podgrafy, pričom v ľavom podgrafe ostávajú interpretácie označené celočíselnou úrovňou (pochádzajúce z rozhodnutia algoritmu Alg. 1.2 na riadkoch 15 až 17) a v pravom podgrafe ostáva konfliktný uzol. Interpretácie z propagácií sa v rôznom pomere rozdeľujú medzi oba podgrafy. Takýmto spôsobom je možné postupným posúvaním rezu smerom nahor a doľava získať ďalšie klauzuly, niektoré z nich sú $\neg S \vee \neg Y$, $\neg X \vee \neg T \vee \neg H$ či $\neg X \vee \neg S \vee \neg G \vee \neg D \vee \neg H \vee \neg I$. V množine možných rezov jedným extrémom je už spomínaný $\neg X \vee \neg Y$, zatiaľ čo opačným extrémom je $\neg R \vee \neg K \vee \neg F \vee \neg C$ obsahujúci iba tie symboly, ktorých interpretácia bola získaná voľbou a nie propagáciou.

VÝBER KONFLIKTNEJ KLAUZULY

Principiálne každá z týchto rezových konfliktných klauzúl môže zabrániť analyzovanému konfliktu. Aby sa mu v budúcnosti predišlo, stačí vybranú klauzulu zaradiť k pôvodným klauzulám riešeného problému. Otázkou však zostáva, ktoré z klauzúl vybrať, pretože takýchto klauzúl je možné generovať veľké množstvo. Tomu zodpovedá aj existencia viacerých schém výberu.

Jedna z najznámejších je založená na tzv. uzloch *UIP* (Unique Implication Point). UIP v implikačnom grafe je taký uzol na aktuálnej úrovni s (na ktorej vznikol konflikt), že každá cesta od uzla s celočíselným označením tejto úrovne ku konfliktnému uzlu vedie cez tento uzol. Ak takýchto uzlov existuje viac, preferuje sa najľavejší (označovaný ako prvý UIP).

V našom prípade na Obr. 1.1 konflikt vznikol na piatej úrovni. Je zrejmé, že existuje iba jeden UIP – $S_{5.5}$. Cez tento uzol prechádza viacero rezov. Jeden z týchto rezov bude pridaný k pôvodným klauzulám ako nová naučená klauzula. Buď to bude prvý nájdený rez prechádzajúci cez UIP, alebo nejaký iný tiež prechádzajúci cez UIP, ktorý je od neho kratší. Je samozrejme možné objaviť všetky takéto rezy a vybrať z nich najkratší (ale počet všetkých rezov môže byť značne veľký), alebo nájsť prvý a tento podrobiť dodatočnej procedúre, v rámci ktorej sa daná klauzula bude minimalizovať (bude sa znižovať počet literálov v nej obsiahnutých).

MINIMALIZÁCIA KONFLIKTNEJ KLAUZULY

V našom prípade, ak sa použije nájdená klauzula $\neg X \vee \neg T \vee \neg H \vee \neg I$, tak je možné dospieť napríklad k rezu $\neg X \vee \neg S \vee \neg D \vee \neg G \vee \neg H \vee \neg I$, ktorý je následne možné postupne zjednodušiť pomocou série rezolvencií

$\neg X \vee \neg S \vee \neg D \vee \neg G \vee \neg H \vee \neg I$	$S \to X$
$\neg S \vee \neg D \vee \neg G \vee \neg H \vee \neg I$	$H \to I$
$\neg S \vee \neg D \vee \neg G \vee \neg H$	$G \wedge E \to H$
$\neg S \vee \neg D \vee \neg G \vee \neg E$	$D \to E$
$\neg S \vee \neg D \vee \neg G$	

DOPLNENIE DPLL O NOVÚ KLAUZULU

Takúto analýzu konfliktov spolu s učením novej klauzuly a jej zaradením k pôvodným klauzulám je možné zaradiť do algoritmu Alg. 1.2 medzi riadky 6 a 7

6.2. $< C, G > := \mathit{conflict\_analysis}(F, I)$
6.3. $C' := \mathit{clause\_minimization}(C, G)$
6.4. $F := F \cup C'$

keď sa najprv vygeneruje konfliktný graf G a na jeho základe sa naučí klauzula C, ktorej splnenie zabráni danému konfliktu, táto klauzula sa následne zjednoduší a zaradí k ostatným klauzulám.

NÁVRAT RIADENÝ KONFLIKTOM

Učenie konfliktných klauzúl môže byť využité aj pre riadenie navracania. Naivná verzia navracania podľa riadkov 7 a 8 algoritmu Alg. 1.2 realizuje návrat vždy iba o jednu úroveň a iba ak to nie je možné, tak sa pokračuje na úroveň ešte pred tým. Túto schému je možné nahradiť niektorou zo schém, reagujúcich na výsledky analýzy konfliktov. Príkladom je schéma založená na intuitívnom chápaní, že ak pre nejaký symbol na úrovni s

nastal konflikt pre obe jeho interpretácie, pričom pre jednu interpretáciu najbližšou príčinou bola voľba na úrovni a a pre druhú interpretáciu zase voľba na úrovni b, nie je nutné sa navrátiť k úrovni $s-1$ ale je možné spätne *skočiť* priamo na úroveň $max(a, b)$.

Uvažujme opäť prípad implikačného grafu podľa Obr. 1.1, z ktorého možno naučiť klauzulu $\neg R \vee \neg K \vee \neg F \vee \neg C$ odvodenú iba z tých interpretácií, ktoré boli určené voľbou ale nie propagovaním. Pretože došlo ku konfliktu, interpretácia symbolu R bola zmenená na $I(R)_5 = FALSE$. Predpokladajme, že následkom toho opäť nastal konflikt a bola odvodená klauzula $R \vee \neg F \vee \neg C$.

Ak by sa algoritmus teraz navrátil k symbolu A, tak po zmene jeho interpretácie $I(A)_4 = TRUE$ na $I(A)_4 = FALSE$ by opäť došlo k skúmaniu vhodnosti interpretácií symbolu R – s už známym výsledkom, pretože odvodené klauzuly signalizujú, že ani jeden z neúspechov nezávisel na interpretácii symbolu A. Preto je možné bezpečne skočiť na voľbu interpretácie toho symbolu, ktorý figuruje v odvodenej klauzule aspoň pre jeden neúspech a je zároveň najbližšie k aktuálnej úrovni (pre náš prípad teda skok spätne na úroveň 3).

DOPLNENIE DPLL O SPÄTNÝ SKOK

Takúto analýzu konfliktov pre určenie cieľa spätného skoku je možné zaradiť do algoritmu Alg. 1.2 medzi riadky 6 a 7

```
6.6.    j := jump_level(G)
6.7.    if flip[s] = 0 then jump := j
6.8.                   else s := max(jump,j)
7.      while s > 0  and flip[s] = 1
8.          s := s − 1
```

keď sa vždy po analýze konfliktného grafu G vygeneruje úroveň, na ktorej bola zvolená tá interpretácia spomedzi interpretácií relevantných voči konfliktu, ktorá je k aktuálnej úrovni najbližšie. Táto úroveň sa bude pamätať. Ak sa jedná už o konflikt pre druhú interpretáciu symbolu na aktuálnej úrovni (teda *flip[s] = 1*), tak sa použije bližšia z úrovní (odpamätanej a aktuálne určenej). Pri potrebe navracania sa najprv skočí na určenú úroveň a až z tejto úrovne v prípade potreby nasleduje navracanie po jednotlivých úrovniach.

VÝBEROVÉ HEURISTIKY

Pri rozširovaní parciálnej interpretácie symbolov (riadok 15 v Alg. 1.2) je tiež možné naivný prístup výberu podľa vopred určeného statického poradia symbolov a priradzovaných pravdivostných hodnôt nahradiť dynamickým prístupom, reagujúcim na priebeh prehľadávania. V tomto smere

existuje veľa *heuristík*. Ako ukážku uveďme zopár používaných heuristík:

- DLIS (Dynamic Largest Individual Sum),
- DLCS (Dynamic Largest Combined Sum),
- MOM (Maximum Occurence on clauses of Minimum size),
- VSIDS (Variable State Independent Decaying Sum).

Heuristika *DLIS* vyberá symbol a k nemu pravdivostnú hodnotu (teda literál) tak, aby voľba umožnila interpretovať čo najviac klauzúl spomedzi tých, ktoré ešte nemajú interpretáciu, ako pravdivých. Je to dynamická heuristika, pretože to, ktoré klauzuly už sú interpretované ako pravdivé (pretože ak je niektorá klauzula interpretovaná ako nepravdivá, tak nasleduje fáza navracania) a ktoré ešte interpretáciu nemajú, závisí od aktuálnej parciálnej interpretácie symbolov. Na jednej strane to je jednoduchá heuristika, na druhej strane však vyžaduje pri každej voľbe kontrolovať všetky klauzuly, čo je výpočtovo náročné. *DLCS* je podobná s tým rozdielom, že výber je rozdelený na dve etapy – najprv na výber symbolu a až potom na výber pravdivostnej hodnoty. Symbol vyberá tak, aby sa vyskytoval v čo najväčšom počte tých klauzúl, ktoré ešte nemajú interpretáciu, bez ohľadu na to či v nich vystupuje v priamom tvare alebo v negovanom tvare. Hodnotu vyberá následne podľa toho, v akom tvare sa vybraný symbol v tých klauzulách vyskytoval častejšie.

HEURISTIKA DLIS

HEURISTIKA DLCS

Heuristika *MOM* k myšlienke interpretácie čo najväčšieho počtu doposiaľ neinterpretovaných klauzúl pridáva ďalšie dve idey: preferenciu krátkych klauzúl (pretože tam je väčšia šanca na vytvorenie jednotkových klauzúl) a preferenciu rovnomerného rozdelenia priameho a nepriameho výskytu (analogicky k hľadaniu v rámci intervalu jeho delením na dve časti – delenie na polovicu je najrýchlejšie v najhoršom prípade). Metrika, ktorú je potrebné maximalizovať pri výbere symbolu je

HEURISTIKA MOM

$$2^k[\#(X) + \#(\neg X)] + \#(X) * \#(\neg X)$$

kde # reprezentuje počítadlo (frekvenciu) výskytu v krátkych klauzulách a k je hodnota zvolená heuristicky. Vyberá sa teda ten symbol X, pre ktorý je táto hodnota najväčšia. Hodnota sa vyberá podľa toho, či je $\#(X) > \#(\neg X)$ (výsledkom je priamy výskyt v literále) alebo naopak (výsledkom je negovaný literál).

Pri *VSIDS* pre každý možný literál každého symbolu existuje počítadlo. Všetky počítadlá sú na začiatku inicializované na nulu. Vždy pri vložení nejakej klauzuly sa inkrementujú počítadlá tých literálov, ktoré sa nachádzajú

HEURISTIKA VSIDS

v tejto vkladanej klauzule. Vždy po určitej dobe sú hodnoty všetkých počítadiel znížené na polovicu. Pri voľbe sa vyberá ten literál (výber sa deje iba medzi literálmi tých symbolov, ktoré ešte neboli interpretované), ktorého počítadlo má aktuálne najvyššiu hodnotu.

Keďže počítadlá sa inkrementujú iba pri vkladaní nových klauzúl (na začiatku to sú klauzuly reprezentujúce problém, neskôr to sú odvodené konfliktné klauzuly), výpočtová náročnosť je menšia. Periodické zmenšovanie hodnôt počítadiel znamená, že sa viac preferujú výskyty v posledne vkladaných klauzulách oproti výskytom v klauzulách vložených skoršie – teda voľba viac reaguje na posledné dianie.

Cvičenia

1. Vytvorte pravdivostnú tabuľku pre nasledovné výrazy vo výrokovom počte:

   a. $\neg A \wedge (A \vee B) \wedge (B \vee C)$
   b. $((\neg A \rightarrow \neg B) \rightarrow A) \rightarrow ((\neg B \rightarrow \neg A) \rightarrow \neg B)$
   c. $(P \rightarrow Q) \wedge (P \rightarrow R) \wedge P \wedge (\neg Q \vee \neg R)$

   Čo sa dá povedať o pravdivosti týchto viet ?

2. Transformujte nasledujúce vety vo výrokovom počte do tvaru CNF:

   a. $A \vee (B \wedge C) \vee (D \wedge E \wedge \neg(A \vee B))$
   b. $\neg(P \rightarrow Q \vee R \leftrightarrow (P \rightarrow Q) \vee (P \rightarrow R))$

3. Pre ilustračný príklad Pr. 1.1 prepíšte znalosti o príklade pomocou výrokového počtu a transformujte ich do tvaru CNF.

4. Prepíšte do výrokového počtu nasledujúce znalosti o piatich priateľoch (Anna, Hedviga, Klára, Rudolf a Vladimír), ktorí radi navzájom on-line komunikujú. Vieme, že platí nasledovné:

   - Alebo Klára alebo Hedviga alebo obidve sú on-line.
   - Rudolf alebo Vladimír sú online, avšak nie obaja.
   - Ak Anna je on-line, potom Rudolf je tiež on-line.
   - Vladimír je on-line vtedy a iba vtedy, keď je on-line Klára.
   - Ak je Hedviga on-line, potom sú online aj Anna a Klára.

   Následne reprezentáciu transformujte do tvaru CNF.

5. Reprezentácie predchádzajúcich dvoch problémov v tvare CNF zakódujte do DIMACS formátu a použite niektorý SAT solver pre zistenie všetkých riešení týchto problémov.

6. Pre výraz v tvare CNF

$$(P \vee Q \vee R) \wedge (P \vee \neg Q \vee \neg R) \wedge (P \vee \neg W) \wedge (\neg Q \vee \neg R \vee \neg W)$$
$$\wedge(\neg P \vee \neg Q \vee R) \wedge (U \vee X) \wedge (U \vee \neg X) \wedge (Q \vee \neg U) \wedge (\neg R \vee \neg U)$$

   nájdite vhodnú interpretáciu symbolov manuálnou simuláciou

   - naivnej verzie algoritmu DPLL podľa Alg. 1.2,
   - ako v predchádzajúcom s pridaním jednotkovej propagácie a propagácie čistého literálu,
   - ako v predchádzajúcom s pridaním učenia nových klauzúl.

   Pri simulácii vytvárajte implikačný graf.

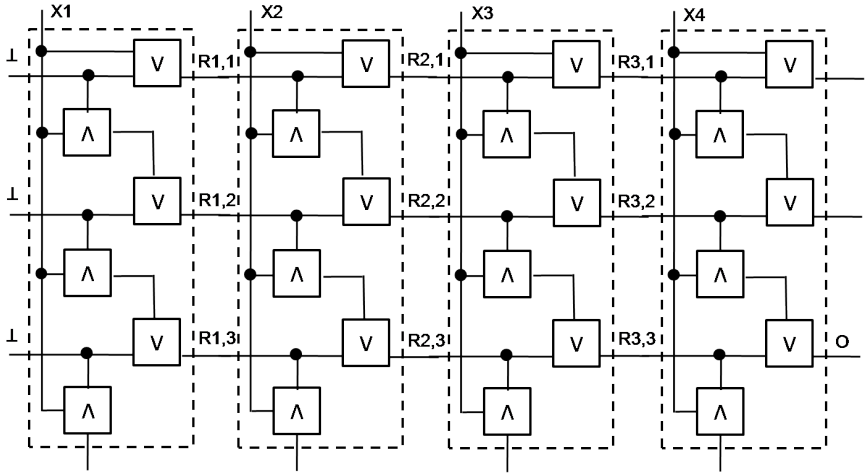
7. Rozvrhnite k dám na šachovnici veľkosti kxk políčok bez toho, aby sa navzájom ohrozovali (problém sa dá vyjadriť pomocou tvrdení "aspoň jedna dáma sa nachádza na X" a "najviac jedna dáma sa nachádza na X", kde pod X sa chápe konkrétny riadok, stĺpec alebo diagonála). Úlohu riešte pre prípad $k = 4$, pričom je potrebné:

   - zvoliť vhodné pomenovanie symbolov tak, aby ich mená korešpondovali s ukotvením symbolov,
   - určiť počet potrebných symbolov a počet klauzúl potrebných pre reprezentáciu,
   - vytvoriť reprezentáciu v DIMACS formáte,
   - nájsť všetky riešenia pomocou SAT solvera.

8. Pripravte rozvrh pre turnaj v golfe, ktorého sa zúčastňuje n ($n = g{*}p$) hráčov, pričom hráči hrajú v g skupinách po p hráčoch. Turnaj trvá w kôl. Podmienkou je, že každá možná dvojica hráčov hrá spolu v jednej skupine najviac jeden krát. Riešte pre usporiadanie hráčov do $g = 5$ skupín po $p = 3$ hráčoch pre rôzne hodnoty w. Použite symboly s významom

   - $X_{i,j,k,l}$ - hráč i hrá na pozícii j v skupine k v kole l
   - $X_{i,k,l}$ - hráč i hrá v skupine k v kole l
   - $X_{i,j,l}$ - hráč i hrá spolu s hráčom j v kole l

9. Formálne verifikujte funkčnosť obvodu zostaveného zo štyroch rovnakých hradiel, ktoré sú spojené podľa nasledovného obrázku



pričom uvažujte iba tie vývody, ktoré sú označené ($R*$ sú vnútorné signály obvodu, $X*$ reprezentujú vstupné signály obvodu, O je výstupným signálom, $\perp$ je signál reprezentujúci logickú nulu).

Kapitola 2

Predikátová logika

Jedným z nedostatkov výrokového počtu je obtiažna reprezentácia komplexnejších konceptov. Napríklad pri vetách typu "ak v Poprade prší, potom v Košiciach svieti slnko" je potrebné zohľadňovať miesta a stavy – a teda každá kombinácia možného miesta a možného stavu musí byť reprezentovaná jedným symbolom, čo vedie jednak k neprehľadnosti a jednak k veľkému počtu použitých symbolov.

Predikátový počet

Naproti tomu *predikátový počet* umožňuje prirodzeným spôsobom reprezentovať vlastnosti objektov a vzťahy medzi objektmi. Reprezentácia uvedenej vety by mohla mať tvar $dazd(poprad) \rightarrow slnko(kosice)$ alebo $poprad(dazd) \rightarrow kosice(slnko)$ podľa toho, či použité predikáty reprezentujú mestá a ich vlastnosti (napr. $dazd(poprad)$ – dážď je vlastnosť Popradu) alebo stavy a ich vlastnosti (napr. $poprad(dazd)$ – miesto výskytu sa stáva vlastnosťou dažďa).

Predikátový počet 0. rádu

Existuje niekoľko navzájom sa líšiacich variantov predikátovej logiky. Najjednoduchší z nich je označovaný ako *predikátová logika nultého rádu*. Syntaktické pravidlá, ktoré určujú čo môžu výrazy predikátového počtu v tomto prípade obsahovať a ako môžu byť kombinované, sú dané BNF (Backus-Naurova Forma) gramatikou v Tab. 2.1.

Predikáty

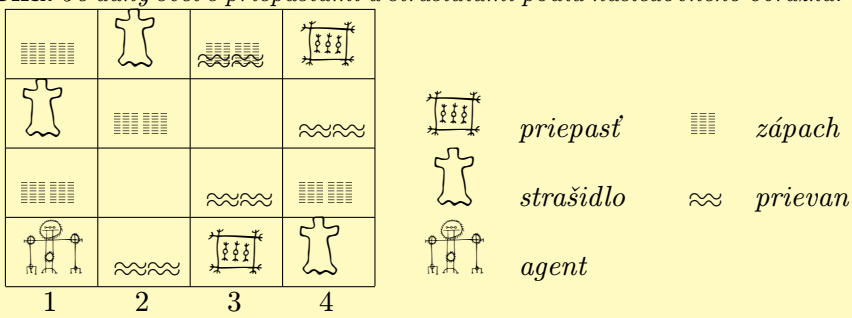
Úlohu symbolov z výrokovej logiky preberajú teraz predikáty – to sú syntakticky najmenšie stavebné prvky, pre ktoré sa určuje ich pravdivosť. Predikát má, na rozdiel od výrokového symbolu, svoju štruktúru – má názov (reprezentovaný predikátovým symbolom) a určitý počet argumentov (označovaný ako *árnosť* predikátu). V už uvedenom zápise predikátu $poprad(dazd)$ je $poprad$ predikátovým symbolom a term $dazd$ zase argumentom daného unárneho predikátu. Ak árnosť predikátu je nulová a teda predikát nemá žiadne argumenty, potom sa označuje ako symbol.

Konštanty a funkcie

Argumentom predikátu môže byť iba term, ktorým je buď konštanta

ILUSTRAČNÝ PRÍKLAD

Úloha: *Je daný svet s priepasťami a strašidlami podľa nasledovného obrázka.*



Tento svet sa riadi podľa nasledujúcich zákonitostí:

1. *strašidlá a samozrejme aj priepasti nemenia svoju polohu,*
2. *ak je na nejakom poli strašidlo, tak na susedných poliach je cítiť zápach,*
3. *ak je na nejakom poli priepasť, tak na susedným poliach je cítiť prievan,*
4. *ak niekto vstúpi na pole s priepasťou, tak zahynie,*
5. *ak niekto vstúpi na pole so strašidlom, tak bude zamordovaný.*

Do tohto sveta bol nasadený agent, ktorý sa v ňom môže pohybovať vertikálnym aj horizontálnym smerom. Je vybavený senzorom zápachu aj senzorom prievanu, ktorých pôsobnosť je však obmedzená iba na to pole, na ktorom sa aktuálne nachádza. Okrem toho je vybavený dostatočnou pamäťou a schopnosťou logicky uvažovať. Jeho úlohou je zmapovať tento svet.

Reprezentácia: *Uvedený problém je reprezentovaný ako logický výraz tvaru:*

$$
\begin{aligned}
& (\forall\, S\ prievan(S) \leftrightarrow \exists\, R\ susedne(S,R) \wedge priepast(R)) \\
\wedge\ & (\forall\, S\ zapach(S) \leftrightarrow \exists\, R\ susedne(S,R) \wedge strasidlo(R)) \\
\wedge\ & (\forall\, A,B,C\ susedne(pole(A,B),pole(A,C)) \leftrightarrow\ po(B,C) \vee po(C,B)) \\
\wedge\ & (\forall\, A,B,C\ susedne(pole(A,B),pole(C,B)) \leftrightarrow\ po(A,C) \vee po(C,A)) \\
\wedge\ & po(1,2) \wedge po(2,3) \wedge po(3,4)
\end{aligned}
$$

Riešenie: *Agent si pri návšteve nejakého poľa do pamäti uloží svoje vnemy, napr. pre pole (1,1) to bude* $\neg prievan(pole(1,1))$ *a* $\neg zapach(pole(1,1))$. *Následne sa pokúsi pre jednotlivé polia dokázať, že sú bezpečné, čo pri pobyte na (1,1) bude* $\neg priepast(pole(2,1))$, $\neg priepast(pole(1,2))$, $\neg strasidlo(pole(2,1))$ *a* $\neg strasidlo(pole(1,2))$. *Ak niektoré pole, na ktorom ešte nebol, je bezpečné, potom prejde naň. Opakovaním tohto postupu si agent vytvorí nasledujúcu mapu skúmaného sveta:*

bezpečné polia:	*(1,1), (2,1), (1,2), (2,2), (2,3), (3,2), (3,3), (3,4), (4,3)*		
priepasť:	*(3,1), (4,4)*	*možná priepasť:*	*(4,2)*
strašidlo:	*(1,3), (2,4)*	*neskúmané polia:*	*(1,4), (4,1)*

Pr. 2.1: Príklad pre reprezentáciu predikátovou logikou

< veta >	::=	< predikát > | < zložená veta >
< predikát >	::=	< predikátový symbol >(< term >*)
< zložená veta >	::=	< unárny operátor > < veta >
	|	< veta > < binárny operátor > < veta >
	|	(< veta >)
< term >	::=	< funkcia > | < konštanta >
< funkcia >	::=	< funkčný symbol >(< term >+)
< unárny operátor >	::=	$\neg$
< binárny operátor >	::=	$\vee \mid \wedge \mid \rightarrow \mid \leftrightarrow$

Tab. 2.1: Syntax predikátového počtu nultého rádu

alebo funkcia. Konštanta sa používa pre reprezentáciu objektu bez zohľadnenia jeho štruktúry. Funkcia, podobne ako predikát, pozostáva z funkčného symbolu a argumentov, ktorých počet udáva árnosť funkcie. Oproti konštante funkcia umožňuje vyjadriť štruktúru objektov alebo objekty, ktoré nie je potrebné reprezentovať samostatným spôsobom ale je možné ich odvodiť z iných objektov. Príkladom konštanty je *dazd*, príkladom funkcie zase *letisko(poprad)*. Pri tvorbe funkcií je možné aj viacnásobné vnorenie rôznych funkcií alebo aj rekurzia tej istej funkcie, napríklad *otec(otec(jano))*.

PREDIKÁTOVÝ POČET 1. RÁDU

Nevýhodou takejto syntaxe je to, že umožňuje reprezentovať iba jednotlivé špecifické objekty, práca so skupinami objektov nie je možná. Pre tento účel *predikátový počet prvého rádu* dopĺňa do svojej syntaxe premenné a kvantifikátory. Tieto doplnenia sú uvedené v Tab. 2.2.

PREMENNÉ

Premenné slúžia na reprezentáciu prvkov nejakej množiny objektov a syntakticky majú rovnakú úlohu ako konštanty a funkcie – môžu byť použité ako argumenty funkcií a predikátov. Tak napríklad v predikáte *dazd(X)*

	...	
< zložená veta >	::=	...
	|	< kvantifikátor > < premenná > < veta >
< term >	::=	... | < premenná >
	...	
< kvantifikátor >	::=	$\exists \mid \forall$

Tab. 2.2: Syntax predikátového počtu prvého rádu (doplnenia voči nultému rádu)

premenná X reprezentuje nejaké miesto s aktuálnym výskytom dažďa.

UNIVERZÁLNY KVANTIFIKÁTOR

Na rozdiel od výrokovej logiky je možné vytvárať všeobecnejšie tvrdenia použitím predikátového počtu prvého rádu. Napríklad to, že každý človek má rád slnko, sa dá vyjadriť ako výraz $\forall X\ cl(X) \rightarrow mr(X, slnko)$, kde predikátový symbol cl vyjadruje vlastnosť "byť človekom" a predikát mr zase vzťah "mať rád". Podobne to, že každý človek má rád všetky zvieratá, by sa vyjadrilo pomocou výrazu $\forall X\ cl(X) \rightarrow (\forall Y\ zv(Y) \rightarrow mr(X, Y))$, kde symbol zv vyjadruje vlastnosť "byť zvieraťom". Znak $\forall$ označuje *univerzálny (zovšeobecňovací) kvantifikátor* reprezentujúci "pre všetky hodnoty premennej". V prípade, že by výraz začínal viacerými kvantifikátormi, potom ich možno skrátene vyjadriť iba jedným, teda namiesto $\forall X\ \forall Y$ je možné použiť $\forall X, Y$ s tým istým významom.

EXISTENČNÝ KVANTIFIKÁTOR

Nie vždy však platí nejaká vlastnosť univerzálne pre všetky objekty z nejakej množiny – napríklad nie každý človek má rád všetky zvieratá alebo každý človek má rád nejaké zviera avšak nie všetky. Toto by sa vyjadrilo pomocou $\exists X\ cl(X) \wedge \exists Y\ zv(Y) \wedge \neg mr(X, Y)$ s prvým významom alebo pomocou $\forall X\ cl(X) \rightarrow \exists Y, Z\ zv(Y) \wedge zv(Z) \wedge mr(X, Y) \wedge \neg mr(X, Z)$ s druhým významom. Znak $\exists$ označuje *existenčný kvantifikátor* reprezentujúci "existuje taká hodnota premennej, pre ktorú platí". Znamená to existenciu aspoň jednej vhodnej hodnoty, čo však nevylučuje, že tých hodnôt môže byť viac vrátane všetkých hodnôt.

Medzi obomi kvantifikátormi existuje vzťah na základe ekvivalentnosti, ktorý umožňuje zámenu jedného kvantifikátora za druhý. Je to

$$\exists X\ < veta > \equiv\ \neg(\forall X \neg < veta >)$$

hovoriaci, že existencia výskytu takej hodnoty, pre ktorú veta platí, je ekvivalentná tomu, že nie je pravda, že veta neplatí pre žiadnu hodnotu danej premennej.

Z toho je zrejmé, že aplikovanie operátora negácie na vetu s kvantifikátorom dáva

$$\neg(\forall X\ p(X)) \equiv \exists X\ \neg p(X) \qquad \neg(\exists X\ p(X)) \equiv \forall X\ \neg p(X)$$

VIAZANÉ A VOĽNÉ PREMENNÉ

Ak sa v logickej vete vyskytuje nejaká premenná a táto premenná je kvantifikovaná pomocou jedného z kvantifikátorov, tak sa označuje ako viazaná premenná. V opačnom prípade sa jedná o voľnú premennú. Pritom je možné, aby premenná hrala naraz obe úlohy. V príklade

$$\forall X\ p(X, Y, Z) \rightarrow \exists Y\ q(X, Y, Z)$$

premenná X je viazaná v celej vete, premenná Z je zase v celej vete voľná a premenná Y je voľná v ľavej časti vety zatiaľ čo v pravej časti je viazaná.

Výroková logika	*Predikátová logika*	*Doména*
symboly	konštanty	objekty
	funkčné symboly	funkcie
	predikátové symboly	relácie

Tab. 2.3: Základné elementy

Voľná premenná je taká, ktorá môže byť nahradená ľubovoľnou inou voľnou premennou s tým, že oba tvary budú ekvivalentné.

INTERPRETÁCIA KONŠTANTY

Na rozdiel od výrokovej logiky, kde interpretácia bola daná pravdivostnými hodnotami symbolov, je teraz potrebné interpretovať konštanty a funkčné ako aj predikátové symboly. Táto interpretácia sa realizuje voči nejakej doméne, ktorá obsahuje objekty, relácie a funkcie (Tab. 2.3). Interpretácia konštanty v predikátovej logike stanovuje, ktorý *objekt* domény (oblasti reálneho sveta) je reprezentovaný danou konštantou. Ak doména obsahuje m_o objektov a existuje n_k konštánt, potom je možných $(m_o)^{n_k}$ interpretácií konštánt. Pritom nie každý objekt domény musí byť reprezentovaný konštantou. Na druhej strane nejaký objekt môže byť reprezentovaný viacerými konštantami. Príkladom je skupina osôb (objekty) ktoré sú známe pod svojimi menami (konštanty). Pritom je možné, že meno niektorej osoby nikto nepoužíva alebo naopak niekto je známy pod viacerými označeniami (nielen meno ale aj prezývky).

INTERPRETÁCIA FUNKČNÉHO SYMBOLU

Interpretácia funkčného symbolu znamená určiť, ktorú *funkciu* domény tento symbol popisuje, pričom pod funkciou domény sa chápe zobrazenie z $\mathcal{D}^n$ do $\mathcal{D}$, kde $\mathcal{D}$ označuje objekty domény a n árnosť funkcie. Interpretácia funkcie je teda daná interpretáciou termov (iba funkcií alebo konštánt) ktoré sú jej argumentami, interpretáciou funkčného symbolu a celá funkcia referuje na nejaký objekt z domény. To sa dá vyjadriť ako

$$f(t_1, ..., t_n)^I = f^I(t_1^I, ..., t_n^I)$$

INTERPRETÁCIA PREDIKÁTOVÉHO SYMBOLU

Podobne interpretácia predikátového symbolu určuje, ktorú *reláciu* domény daný symbol popisuje, pričom pod reláciou domény sa chápe zobrazenie z $\mathcal{D}^n$ do množiny $\{TRUE, FALSE\}$. A teda interpretácia predikátu je daná interpretáciou termov (opäť iba konštanty alebo funkcie), interpretáciou predikátového symbolu a celému predikátu je priradená pravdivostná hodnota. Predikát je pravdivý v nejakej interpretácii I, ak tá relácia domény, na ktorú odkazuje predikátový symbol v I, platí medzi tými objektami domény,

na ktoré odkazujú argumenty predikátu v danej interpretácii I. V prípade, že árnosť predikátu je nulová a teda predikát je symbolom, potom je interpretovaný rovnako ako symbol vo výrokovej logike – interpretácia mu priamo priradí jednu z pravdivostných hodnôt.

INTERPRETÁCIA VETY

Pre ľubovoľnú vetu F a ľubovoľnú interpretáciu I, pravdivostná hodnota F^I, ktorá je priradená vete F interpretáciou I sa určí rekurzívnym spôsobom:

- ak F je predikát, potom $F^I = I(F)$
- ak $F = (\neg G)$, tak $F^I = \neg(G^I)$
- ak $F = (G \odot H)$ a $\odot$ reprezentuje jeden z definovaných binárnych operátorov, tak $F^I = (G^I) \odot (H^I)$

kde $I(F)$ označuje určenie pravdivostnej hodnoty predikátu na základe jeho interpretácie (interpretácie predikátového symbolu ako aj všetkých termov vystupujúcich ako jeho argumenty).

Pravdivosť vety je teda daná pravdivosťou predikátov a predpismi pre odvádzanie výslednej pravdivostnej hodnoty, platnými pre jednotlivé operátory (podľa pravdivostných tabuliek v Tab. 1.2).

INTERPRETÁCIA PREMENNEJ

V prípade výskytu premenných sa situácia trochu komplikuje. Za predpokladu, že vo vete vystupujú iba viazané premenné, tak sa vytvárajú *rozšírené interpretácie*, kde v každej je premenná interpretovaná ako jeden z objektov, na ktoré premenná odkazuje. Potom

- všeobecne kvantifikovaná veta je pravdivá v interpretácii I, ak je pravdivá pre všetky rozšírené interpretácie odvodené z I,
- existenčne kvantifikovaná veta je pravdivá v interpretácii I, ak je pravdivá pre aspoň jednu rozšírenú interpretáciu odvodenú z I.

Univerzálny kvantifikátor teda umožňuje formulovať tvrdenia o každom objekte, zatiaľ čo existenčný kvantifikátor umožňuje formulovať tvrdenia o nejakom objekte bez toho, aby tento objekt bol pomenovaný.

Pre ilustráciu predpokladajme existenciu domény $\mathcal{D}$, ktorá obsahuje štyri objekty: *fero*, *jano*, *ondro* a *gusto*. Navyše obsahuje ešte jednu binárnu a jednu unárnu reláciu. Binárna relácia (označovaná *nema_rad* s významom, že objekt, ktorý je prvým argumentom, neobľubuje objekt, ktorý je druhým argumentom) je pravdivá pre dvojice objektov *jano-ondro*, *ondro-jano*, *fero-ondro*, *ondro-fero*, *jano-gusto* a *gusto-jano*. Pre ostatných desať dvojíc je relácia nepravdivá. Unárna relácia (označovaná *kamarat*) je pravdivá pre objekty *gusto* a *ondro*, pre ostatné dva objekty je nepravdivá.

Skúsme vyjadriť tvrdenie "všetci nepriatelia môjho nepriateľa sú moji priatelia". Nech je to reprezentované vetou

$$\forall X \; nepriatel(moj\_nepriatel, X) \rightarrow priatel(X)$$

s jednou konštantou a dvomi rôznymi predikátmi. Pre interpretáciu konštanty sú v danej doméne štyri rôzne možnosti, predikátový symbol *nepriatel* má iba jednu možnú interpretáciu a predikátový symbol *priatel* tiež iba jednu. Jedna možná interpretácia (párujúca elementy predikátového počtu s elementmi domény) je

$$moj\_nepriatel = fero,\; nepriatel = nema\_rad,\; priatel = kamarat$$

ktorej prislúchajú štyri rozšírené interpretácie podľa Tab. 2.4. Z tabuľky

X	nepriatel(fero,X)	priatel(X)	nepriatel(fero,X) → priatel(X)
fero	FALSE	FALSE	TRUE
jano	FALSE	FALSE	TRUE
ondro	TRUE	TRUE	TRUE
gusto	FALSE	TRUE	TRUE

Tab. 2.4: Pravdivostná tabuľka pre rozšírené interpretácie

je zrejmé, že pre danú interpretáciu je celá veta interpretovaná ako pravdivá, lebo je pravdivá pre každú rozšírenú interpretáciu. Podobne to je aj pre prípad, že konštanta *moj_nepriatel* je interpretovaná ako objekt *jano*. Na druhej strane, ak by táto konštanta bola interpretovaná či už ako *ondro* alebo *gusto*, tak celková veta by bola interpretovaná ako nepravdivá, pretože v týchto prípadoch by bola nepravdivá aspoň v jednej rozšírenej interpretácii.

Počet interpretácií

Jednotlivé interpretácie sa teda líšia v tom, ako interpretujú konštanty a symboly voči prvkom domény. Ak veta obsahuje n_k konštánt, n_f^i funkčných symbolov árnosti i a n_p^i predikátových symbolov árnosti i, a doména obsahuje zodpovedajúce počty m_o objektov, m_f^i funkcií árnosti i a m_p^i relácií i-tej árnosti medzi objektami, potom počet všetkých možných interpretácií je daný ako

$$(m_o)^{n_k} \left(\prod_{i=1,\ldots} (m_f^i)^{n_f^i} \right) \left(\prod_{i=1,\ldots} (m_p^i)^{n_p^i} \right)$$

kde $\prod_{i=1,\ldots}$ označuje súčin.

Ak by doména obsahovala napríklad ešte jednu unárnu reláciu (označovanú *sportovec*), ktorá by bola pravdivá iba pre objekt *fero*, tak by boli možné ďalšie štyri rozšírené interpretácie, ktoré by obsahovali *priatel* = *sportovec* (pri všetkých štyroch by bola daná veta interpretovaná ako nepravdivá).

INTERPRETÁCIA PODĽA SYNTAKTICKEJ ZHODY

Je zrejmé, že interpretácia *priatel* = *kamarát* je lepšia než *priatel* = *športovec* – avšak logika je formálny systém, ktorý nepracuje so sémantikou označení relácií a funkcií. Z toho dôvodu je vhodné (ak to je možné) v reprezentovaných vetách používať symboly rovnakého tvaru ako sú názvy zodpovedajúcich relácií a funkcií v uvažovanej doméne. Potom je možné uvažovať iba tie interpretácie, kde funkčné symboly a predikátové symboly sú interpretované na doménové funkcie a relácie s rovnakými názvami (interpretácia založená na *syntaktickej zhode mien*).

Bolo by to vhodné aj pri konštantách (ak by sme napríklad vedeli, ktorý konkrétny objekt spĺňa predstavu "môjho nepriateľa"). Nie vždy to je možné – niekedy proste vopred nie je známa vhodná interpretácia a nezostáva nič iné ako použiť konštantu, ktorej meno nezodpovedá priamo žiadnemu objektu z domény (príkladom takéhoto prístupu je konštanta *moj_nepriatel*) alebo použiť premennú, ak je možnosť jej vhodnej kvantifikácie. Na základe tohto by sa dala veta "všetci nepriatelia môjho nepriateľa sú moji priatelia" pre danú doménu (s ohľadom na mená v nej použité) pretransformovať do tvaru

$$\exists Y\ \neg kamarat(Y) \wedge (\forall X\ nema\_rad(Y, X) \rightarrow kamarat(X))$$

kde namiesto konštanty sa použila forma premennej s vlastnosťou, ktorá ju definuje.

REPREZENTÁCIA PRE ILUSTRAČNÝ PRÍKLAD

Podobne aj v ilustračnom príklade Pr. 2.1 v reprezentácii problému sa používajú rovnaké názvy ako v slovnom popise toho problému (*strasidlo*, *priepast*, *zapach*, *prievan*, atď.).

Medzi pozíciou, ktorá je reprezentovaná jedným poľom v danom svete, a umiestnením tejto pozície podľa dvoch súradníc existuje závislosť, ktorú je možné modelovať ako štrukturálny vzťah medzi príslušnými objektmi. Vďaka tomuto nie je potrebné uvažovať jednotlivé polia a hodnoty súradníc ako samostatné objekty, ale je možné jeden typ objektu odvodiť z iného typu objektov. Preto polia nepoužívajú explicitné pomenovania ale sú vyjadrené binárnou funkciou s príslušným funkčným symbolom.

Takto implicitne vyjadrené polia sú nositeľmi vlastností: *priepast* (na danom poli sa nachádza priepasť), *strasidlo* (na danom poli je umiestnené strašidlo), *prievan* (agent na danom poli registruje prievan) a *zapach* (agent na danom poli registruje zápach). To, či nejaké pole má tú ktorú vlastnosť, je reprezentované príslušným unárnym predikátom.

V popise zákonitostí ilustračného sveta sa vyskytuje fráza "nejaké pole". Keďže nie je jasné, o ktoré konkrétne pole sa jedná, bola zvolená reprezentácia pomocou premennej. Keďže je domnienka, že vlastnosti č. 2 (ak je na nejakom poli strašidlo, tak na susedných poliach je cítiť zápach) a č. 3 (ak je na nejakom poli priepasť, tak na susedných poliach je cítiť prievan) sú všeobecne platné pre každé pole, tak príslušná premenná je univerzálne kvantifikovaná.

Medzi poliami existujú závislosti, keď vlastnosť jedného poľa súvisí s vlastnosťou susedného poľa (napr. *priepast* a *prievan*). Ak by sa z výskytu priepasti na nejakom poli odvádzal prievan na susednom poli, tak potom susedné pole by bolo reprezentované premennou s univerzálnou kvantifikáciou (pretože daná vlastnosť platí pre každé zo susedných polí). Naopak, ak by sa z prievanu na nejakom poli odvádzal výskyt priepasti na susednom poli, tak potom by susedné pole bolo reprezentované premennou s existenčnou kvantifikáciou (vlastnosť nemusí platiť pre všetky susedné polia, avšak platí aspoň pre jedno z nich).

REPREZENTÁCIA SUSEDNOSTI

Na záver je ešte potrebné vyjadriť pojem susednosti polí, ktorý je založený na následnosti dvoch čísel. Keďže v predikátovej logike žiadna vlastnosť následnosti či usporiadania čísel nie je definovaná, je ju nutné explicitne reprezentovať pomocou predikátu, definujúceho úplné usporiadanie objektov, vyjadrujúcich hodnoty súradníc. Použitý predikát *po* vyjadruje, ktorý objekt nasleduje za ktorým objektom, teda napríklad *po(1,2)* reprezentuje fakt, že po čísle 1 nasleduje číslo 2.

$<$ veta $>$	::=	$<$ klauzula $> \mid <$ klauzula $> \wedge <$ veta $>$
$<$ klauzula $>$	::=	$<$ literál $> \mid <$ literál $> \wedge <$ klauzula $>$
$<$ literál $>$	::=	$<$ predikát $> \mid \neg <$ predikát $>$
$<$ predikát $>$	::=	$<$ predikátový symbol $>(<$ term $>^*)$
$<$ term $>$	::=	$<$ konštanta $>$
	$\mid$	$<$ funkcia $>$
	$\mid$	$<$ premenná $>$
$<$ funkcia $>$	::=	$<$ funkčný symbol$>(<$ term $>+)$

Tab. 2.5: CNF pre predikátovú logiku

CNF PRE PREDIKÁTOVÚ LOGIKU

Aj v predikátovej logike sa používa CNF. Jej definícia pre oblasť predikátovej logiky je v Tab. 2.5. Opäť je to konjunkcia disjunkcií literálov. Avšak teraz literál namiesto symbolu obsahuje predikát buď v priamej alebo negovanej podobe. Ako vidno, transformácia do takejto formy musí nielen nahradiť binárne operátory s výnimkou $\wedge$ a $\vee$ (a tie usporiadať tak, aby sa

vstup: veta F v ľubovoľnom tvare
výstup: veta F v CNF tvare

1. $F :=$ *eliminácia_ekvivalencií(F)*
2. $F :=$ *eliminácia_implikácií(F)*
3. $F :=$ *vnorenie_negácií(F)*
4. $F :=$ *štandardizácia_premenných(F)*
5. $F :=$ *vylúčenie_existenčných_kvantifikátorov(F)*
6. $F :=$ *vylúčenie_zovšeobecňovacích_kvantifikátorov(F)*
7. $F :=$ *úprava_na_CNF(F)*

Alg. 2.1: Transformácia do CNF

disjunkcia vyskytovala v rámci konjunkcie a nie naopak) a negáciu vnoriť na úroveň literálov, ale aj odstrániť oba typy kvantifikátorov.

PREVOD DO CNF

Ľubovoľná veta v predikátovom počte prvého rádu môže byť transformovaná na ekvivalentnú vetu vyjadrenú v tvare CNF. Algoritmus pre túto transformáciu pozostáva zo siedmich krokov (Alg. 2.1), pričom štyri z nich (kroky 1, 2, 3 a 7) majú rovnaký cieľ a sú rovnako realizovateľné ako pri výrokovom počte.

ŠTANDARDIZÁCIA PREMENNÝCH

Cieľom štandardizácie premenných je zabezpečiť, aby žiadne dve rôzne premenné neboli pomenované rovnako. Ilustrujme to na dvoch jednoduchých ukážkach

$$(\forall X\ p(X)) \vee (\exists X\ q(X)) \qquad \forall X\ (p(X) \rightarrow \exists X\ q(X))$$

kde v oboch prípadoch premenná X, hrajúca úlohu argumentu predikátu p, je rôzna od premennej X, ktorá vystupuje ako argument predikátu q. Keďže sa nepoužívajú voľné premenné, tak každá premenná je kvantifikovaná a teda by jej mal zodpovedať práve jeden kvantifikátor, ktorý ju uvádza. V prípade, že pre jedno meno premennej existuje viac kvantifikátorov, potom viac premenných používa rovnaké meno – toto meno v rámci pôsobnosti nejakého kvantifikátora označuje nejakú premennú, zatiaľ čo v rámci pôsobnosti iného kvantifikátora označuje inú premennú. Ak sa premenná nachádza v rámci pôsobnosti viacerých kvantifikátorov (teda kvantifikátory sú vnorené), tak premenná prináleží k tomu kvantifikátoru, ktorý je k

nej bližšie. Keďže na mene premennej nezáleží a platí, že dve vety, ktoré sa líšia iba názvami premenných, sú ekvivalentné, tak pre účely rozoznávania premenných je potrebné ich premenovať tak, aby žiadne dve premenné nepoužívali rovnaké meno. Naše ukážky sa tak môžu transformovať napríklad na tvar

$$(\forall X\ p(X)) \vee (\exists Y\ q(Y)) \qquad \forall Z\ (p(Z) \rightarrow \exists X\ q(X))$$

Vylúčenie ∃ kvantifikátora

Výskyt existenčného kvantifikátora vo výraze $\exists X\ kamarat(X)$ hovorí, že existuje aspoň jeden taký objekt v doméne $\mathcal{D}$, pre ktorý je unárny predikát *kamarat* pravdivý (z už uvedeného príkladu tejto domény na strane 30 vieme, že to sú objekty *ondro* a *gusto*). To znamená, že pravdivosť výrazu sa zachová, ak by sme premennú X nahradili konštantou, ktorá je interpretovaná na jeden z týchto objektov – napríklad by sa použil tvar $kamarat(ondro)$

Iná situácia by bola, ak by sme nevedeli, pre ktorý z objektov je daný predikát pravdivý – jediné čo vieme je to, že taký objekt existuje. Je možné tento neznámy objekt pomenovať pomocou novej konštanty, ktorá sa doposiaľ nevyskytla (aby nedošlo ku konfliktu s nejakou už existujúcou konštantou), a všetky výskyty existenčne kvantifikovanej premennej nahradiť touto konštantou. Takto je možné výraz $\exists X\ kamarat(X)$ nahradiť pomocou $kamarat(niekto)$, kde *niekto* je nová konštanta. Takáto konštanta sa označuje ako *skolemova konštanta* a celý proces náhrady sa nazýva *skolemizácia*.

Skolemova konštanta

Zložitejšia situácia nastáva, ak existenčný kvantifikátor sa nachádza v oblasti pôsobenia zovšeobecňovacieho kvantifikátora ako pri

$$\forall Y\ \exists X\ kamarat(X) \vee nema\_rad(Y, X)$$

Skolemova funkcia

pretože teraz sa náhrada premennej X deje v kontexte, ktorý je daný univerzálne kvantifikovanou premennou Y. Preto sa namiesto skolemovej konštanty použije *skolemova funkcia*, ktorá je funkciou tej premennej, ktorá vytvára kontext[1]. Podobne ako pri konštante, teraz funkčný symbol nesmie vytvárať kolízie. Pri takejto náhrade by bol výsledok

$$\forall Y\ kamarat(niekto(Y)) \vee nema\_rad(Y, niekto(Y))$$

Môže nastať prípad, keď existenčný kvantifikátor je v oblasti pôsobenia viacerých zovšeobecňovacích kvantifikátorov, ako napríklad

$$\forall Z\ \forall Y\ \exists X\ \neg nema\_rad(Z, X) \vee \neg nema\_rad(X, Y)$$

[1] Skolemovu konštantu možno chápať ako skolemovu funkciu s nulovou árnosťou.

V takomto prípade skolemova funkcia bude mať viac argumentov – pre daný prípad premenná X bude nahradená binárnou funkciou $niekto(Z, Y)$.

VYLÚČENIE ∀ KVANTIFIKÁTORA

Zovšeobecňovacie kvantifikátory je možné posunúť úplne na začiatok vety. Umožňujú to tieto pravidlá (predpokladá sa, že predikát q neobsahuje premennú X v rámci svojich argumentov)

$$\begin{aligned}
(\forall X\ p(X,...)) \vee q(...) &\rightarrow \forall X\ (p(X,...) \vee q(...)) \\
q(...) \vee (\forall X\ p(X,...)) &\rightarrow \forall X\ (q(...) \vee p(X,...)) \\
(\forall X\ p(X,...)) \wedge q(...) &\rightarrow \forall X\ (p(X,...) \wedge q(...)) \\
q(...) \wedge (\forall X\ p(X,...)) &\rightarrow \forall X\ (q(...) \wedge p(X,...))
\end{aligned}$$

Po vykonaní posunu je výsledkom tvar, kde všetky kvantifikátory sú nasledované konjunkciou klauzúl, ktorá kvantifikátory neobsahuje. A keďže všetky premenné, ktoré sa vo vete vyskytujú, sú všeobecne kvantifikované, je možné kvantifikátory jednoducho odstrániť a ponechať iba zvyšnú časť vety.

Ako príklad pre transformáciu do CNF uvažujme nasledujúcu časť reprezentácie z ilustračného príkladu Pr. 2.1

$$\forall\ S\ prievan(S) \leftrightarrow (\exists\ R\ susedne(S,R) \wedge priepast(R))$$

Z tejto vety po odstránení operátorov ekvivalencie a implikácie získame

$$\begin{aligned}
\forall\ S\quad & (\neg prievan(S) \vee (\exists\ R\ susedne(S,R) \wedge priepast(R))) \ \wedge \\
& (\neg(\exists\ R\ susedne(S,R) \wedge priepast(R)) \vee (prievan(S)))
\end{aligned}$$

a po vnorení negácie bude mať veta nasledovný tvar

$$\begin{aligned}
\forall\ S\quad & (\neg prievan(S) \vee (\exists\ R\ susedne(S,R) \wedge priepast(R))) \ \wedge \\
& ((\forall\ R\ \neg susedne(S,R) \vee \neg priepast(R)) \vee (prievan(S)))
\end{aligned}$$

Pretože dve rôzne premenné kvantifikované dvomi kvantifikátormi používajú rovnaké meno, je potrebné jednu z nich premenovať

$$\begin{aligned}
\forall\ S\quad & (\neg prievan(S) \vee (\exists\ R\ susedne(S,R) \wedge priepast(R))) \ \wedge \\
& ((\forall\ Q\ \neg susedne(S,Q) \vee \neg priepast(Q)) \vee (prievan(S)))
\end{aligned}$$

Vo vete sa vyskytuje jedna existenčne kvantifikovaná premenná. Túto premennú je potrebné nahradiť pomocou skolemovej funkcie, čo umožní odstránenie existenčného kvantifikátora

$$\begin{aligned}
\forall\ S\quad & (\neg prievan(S) \vee (susedne(S, f(S)) \wedge priepast(f(S)))) \ \wedge \\
& ((\forall\ Q\ \neg susedne(S,Q) \vee \neg priepast(Q)) \vee (prievan(S)))
\end{aligned}$$

Po presunutí zovšeobecňovacích kvantifikátorov je možné ich odstrániť a záverečne ešte vetu upraviť na tvar CNF

$$\begin{aligned} & (\neg prievan(S) \vee susedne(S, f(S))) \\ \wedge \quad & (\neg prievan(S) \vee priepast(f(S))) \\ \wedge \quad & (\neg susedne(S, Q) \vee \neg priepast(Q) \vee prievan(S)) \end{aligned}$$

ROZŠIROVANIE BÁZY ZNALOSTÍ

Všetky klauzuly, ktoré boli odvodené pre reprezentáciu nejakej úlohy, vytvárajú spolu znalostnú bázu KB danej úlohy (teda znalostná báza má tvar konjunkcie klauzúl). Na základe tejto znalostnej bázy je možné *dedukovať* ďalšie znalosti[2], opäť vo forme klauzúl, ktoré je následne možné do tejto bázy pridať. Celý proces sa tak môže opakovať až dovtedy, keď už deduktívne odvodzovanie nie je schopné vyprodukovať znalosť, ktorá by ešte nebola známa.

Tak napríklad v ilustračnom príklade Pr. 2.1 je možné na základe vnemov agenta, stojaceho na pozícii $pole(1,1)$ do znalostnej bázy pridať nové znalosti $\neg prievan(pole(1,1))$ a $\neg zapach(pole(1,1))$ a na základe toho odvodiť $\neg priepast(pole(1,2))$, $\neg priepast(pole(2,1))$, $\neg strasidlo(pole(1,2))$ a $\neg strasidlo(pole(2,1))$. Po presune agenta na bezpečnú pozíciu $pole(2,1)$ sa zaznamená vnem agenta, získaný na tomto poli, ako $prievan(pole(2,1))$ a $\neg zapach(pole(2,1))$ a následne sa odvodí neprítomnosť strašidla v okolí. Po presune na ďalšiu známu bezpečnú pozíciu $pole(1,2)$ a zaznamenaní vnemov je už možné odvodiť prítomnosť strašidla na pozícii $pole(1,3)$ a súčasne neprítomnosť priepasti na tomto poli, prítomnosť priepasti na pozícii $pole(3,1)$ a súčasne neprítomnosť strašidla na tomto poli a bezpečnosť pozície $pole(2,2)$. Takže agent sa môže presunúť na ďalšie pole, o ktorom sa vie že je bezpečné, a jeho skúmanie sveta môže pokračovať.

LOGICKÉ VYPLÝVANIE

Odvodzovanie je založené na pojme logického vyplývania. To, že nejaká veta α zahŕňa nejakú inú vetu β (značené ako $\alpha \models \beta$) znamená, že v každej interpretácii, v ktorej je α pravdivá, je súčasne pravdivá aj β (ale pravdivosť β nie je obmedzená iba na tieto interpretácie, pretože β môže byť pravdivá aj v takej interpretácii, kde α pravdivá nie je). V pravdivosti vety α je súčasne zahrnutá aj pravdivosť vety β respektíve z nej vychádza. Je zrejmé, že úlohu α hrá znalostná báza KB obsahujúca známe informácie a úlohu β zase nová znalosť, ktorá môže byť odvodená z KB.

Dôkaz zahrnutia jednej vety druhou je možné v zásade realizovať tromi rôznymi spôsobmi:

- kontrolou interpretácií,

[2] Prostredníctvom riešenia dedukčnej úlohy ako už bolo spomenuté v kapitole venovanej výrokovej logike.

- priamym dôkazom,
- dôkazom sporom.

KONTROLA INTERPRETÁCIÍ

Pri prvom spôsobe je potrebné skontrolovať všetky možné interpretácie, vybrať z nich tie, pri ktorých je α interpretovaná ako pravdivá a skontrolovať, či v každej takejto interpretácii je aj β interpretovaná ako pravdivá. Jedná sa o ideovo jednoduchý spôsob, avšak z pohľadu použitia je vďaka veľkému množstvu možných interpretácií táto možnosť viac teoretická než praktická.

PRIAMY DÔKAZ

Pri *priamom dôkaze* sa používajú rozličné odvodzovacie pravidlá (napr. modus ponens), umožňujúce transformovať logické vety na ekvivalentné tvary. Dôkaz má potom podobu výberu a uplatňovania takýchto pravidiel na vetu α dovtedy, až sa táto zo svojej počiatočnej podoby pretransformuje na tvar, ktorý bude obsahovať β. Pri tomto spôsobe je problematickou realizácia voľby, ktoré pravidlo použiť na ktorú časť vety v ktorom transformačnom kroku. Keďže táto voľba sa obtiažne realizuje "cieľovo orientovaným" spôsobom, počet možností nefavorizuje ani tento spôsob.

DÔKAZ SPOROM

Dôkaz sporom je založený na dvoch pojmoch: validnosti a splniteľnosti. Ich význam je pre predikátovú logiku rovnaký ako pre logiku výrokovú:

- validnosť – veta je validná vtedy, ak je pravdivá pri každej interpretácii,
- splniteľnosť – veta je splniteľná vtedy, ak existuje aspoň jedna taká interpretácia, pri ktorej je veta pravdivá. Ak žiadna taká interpretácia neexistuje, potom veta je nesplniteľná.

Na základe využitia týchto pojmov je možné pre dôkaz platnosti nejakej vety β (označovanej aj ako cieľová znalosť) odvodiť

$$\begin{aligned} KB \models \beta &\leftrightarrow valid(KB \rightarrow \beta) \\ valid(KB \rightarrow \beta) &\leftrightarrow valid(\neg KB \vee \beta) \\ valid(\neg KB \vee \beta) &\leftrightarrow nespl(\neg(\neg KB \vee \beta)) \\ nespl(\neg(\neg KB \vee \beta)) &\leftrightarrow nespl(KB \wedge \neg\beta) \end{aligned}$$

kde *valid(F)* znamená, že veta F je validná, a *nespl(F)* zase že veta F je nesplniteľná. Z uvedeného vyplýva, že pre dôkaz platnosti nejakej novej cieľovej znalosti na základe danej znalostnej bázy (teda že znalostná báza KB zahŕňa cieľovú znalosť resp. že cieľová znalosť logicky vyplýva zo znalostnej bázy) stačí dokázať, že súčasne nemôže platiť znalostná báza a aj negácia cieľovej znalosti. Znalostná báza (reprezentovaná ako konjunkcia klauzúl)

sa jednoducho rozšíri pridaním jednej alebo viacerých klauzúl, reprezentujúcich negáciu dokazovanej cieľovej znalosti. Ak sa dokáže neplatnosť takto rozšírenej znalostnej bázy (teda pridaná negácia cieľovej znalosti je v rozpore so známymi axiómami reprezentovanými pôvodnou znalostnou bázou), tak to znamená platnosť cieľovej znalosti.

Rezolvencia

Pri tomto dôkaze sa používa *rezolvenčná metóda*, ktorá je populárnou metódou v oblasti automatického dokazovania. Jej popularita je založená na tom, že môže byť ľahko automatizovaná, pretože je založená na používaní iba jedného odvodzovacieho pravidla – *rezolvenčného odvodzovacieho pravidla*. Toto pravidlo má tvar

$$\frac{P \vee Q \qquad \neg Q \vee R}{P \vee R}$$

Komplementárne literály

kde vstupom sú dve klauzuly ľubovoľnej dĺžky (v uvedenom prípade to sú klauzuly $P \vee Q$ a $\neg Q \vee R$), pričom dĺžky oboch klauzúl nemusia byť rovnaké. Tieto vstupné klauzuly musia obsahovať dvojicu *komplementárnych literálov* (každý z nich pochádza z inej vstupnej klauzuly). Na rozdiel od výrokovej logiky, kde úlohu komplementárnych literálov hrali symbol a jeho negácia, pri predikátovej logike sa jedná o literály založené na predikátoch – jeden z nich je vždy priamo vyjadreným literálom a druhý sa zhoduje[3] s negáciou prvého literálu.

Rezolventa

Výsledkom je tretia klauzula (v uvedenom prípade $P \vee R$), ktorá sa nazýva *rezolventa*. Obsahuje všetky literály oboch vstupných klauzúl s výnimkou komplementárnej dvojice literálov. Ak jedna zo vstupných klauzúl má jednotkovú dĺžku (obsahuje iba jeden literál), hovorí sa o *jednotkovej rezolvencii*. Vtedy dĺžka rezolventy je o jednotku menšia ako dĺžka dlhšej zo vstupných klauzúl – dochádza ku skracovaniu dĺžok. Ak obe vstupné klauzuly obsahujú väčší počet literálov bez toho, aby sa nejaký literál vyskytoval v oboch vstupných klauzulách, potom výsledná rezolventa obsahuje viac literálov než dlhšia zo vstupných klauzúl – dochádza k zväčšovaniu dĺžok.

Detekcia sporu

Ak by sa rezolvencia vykonávala nad množinou klauzúl, ktoré sú navzájom protirečivé, tak po nejakom počte opakovaných rezolvencií bude detekovaný spor. To možno ilustrovať napríklad na klauzulách $\neg P \vee Q$, $\neg Q \vee R$, P a $\neg R$. V tomto prípade by rezolvencie mohli vyzerať

$$\dfrac{\dfrac{\dfrac{\neg P \vee Q \qquad \neg Q \vee R}{\neg P \vee R} \qquad P}{R} \qquad \neg R}{\square}$$

[3] Čo sa myslí pod "zhodovaním" bude vysvetlené neskôr.

kde symbol □ reprezentuje prázdnu klauzulu, ktorá znamená výskyt *sporu*. V prípade, že už nie je možné vytvoriť takú rezolventu, ktorá ešte nebola vytvorená, a spor ani v jednom prípade nenastal, tak nie je možné vyvrátiť platnosť negácie cieľovej hypotézy a teda nie je možné za daných okolností dokázať platnosť skúmanej cieľovej hypotézy.

Ak by vstupné klauzuly obsahovali rovnaký literál, tak potom by rezolventa obsahovala dva výskyty tohto literálu. V tomto prípade je potrebné rezolventu previesť na ekvivalentný tvar použitím $P \vee P \equiv P$. Ak by sa to neurobilo, tak by mohla nastať situácia podľa

$$\dfrac{\dfrac{P \vee Q \qquad \neg Q \vee P}{P \vee P} \qquad \neg \mathrm{P}}{P}$$

kedy nedochádza k detekcii sporu.

SÉMANTIKA REZOLVENČNÉHO PRAVIDLA

Zmysel rezolvenčného odvodzovacieho pravidla nie je na prvý pohľad najzreteľnejší – situácia sa zmení, ak pravidlo bude vyjadrené v ekvivalentnom tvare

$$\dfrac{\neg P \to Q \qquad Q \to R}{\neg P \to R}$$

čo je vlastne definíciou *tranzitívnosti* operácie implikácie. Keď sa zoberie do úvahy aj *pravidlo kontrapozície*

$$X \to Y \equiv \neg Y \to \neg X$$

tak sa dá rezolvenčné odvodzovacie pravidlo transformovať na ekvivalentné pravidlo

$$\dfrac{\neg P \to Q \qquad Q \to R}{\neg R \to P}$$

ktoré vyjadruje podstatu dôkazu sporom – ak sa vyjde z negácie cieľovej hypotézy $\neg P$, tak na základe tranzitívnej aplikácie pravidla modus ponens by nejaká znalosť R mala byť pravdivá. Ak sa však zistí, že táto znalosť je v skutočnosti nepravdivá a platí $\neg R$ (a teda došlo k sporu medzi R a $\neg R$), tak z toho sa priamo odvodí platnosť cieľovej hypotézy P.

ZHODA LITERÁLOV

Pri zavedení rezolvenčného odvodzovacieho pravidla bol použitý pojem "zhodovania sa" pri definícii komplementárnych literálov. Otázkou je, čo tento pojem vlastne reprezentuje. Odpoveď závisí od toho, aká logika bola použitá pre reprezentáciu znalostí. Ak sú použité iba predikátové symboly (teda árnosť všetkých predikátov je nulová)[4], potom pod zhodnosťou sa

[4]Keďže takáto logika je ekvivalentná výrokovej logike, tak rezolvenčnú metódu možno použiť aj pri riešení dedukčnej úlohy pri reprezentácii výrokovým počtom.

myslí rovnaké pomenovanie symbolu. V tomto prípade pod komplementárnymi literálmi sa rozumie dvojica predikát a jeho negácia.

Zložitejšia situácia nastáva, keď v predikátoch sú použité aj termy vrátane premenných. V takomto prípade intuitívne je možné za komplementárne literály považovať trebárs dvojicu $p(q, f(r))$ a $\neg p(q, f(r))$, zatiaľ čo na druhej strane zase dvojica $p(s, g(r))$ a $\neg p(g(s), r)$ nebude považovaná za komplementárne literály. Avšak čo napríklad pri dvojici $p(X, f(X))$ a $\neg p(g(Z), Y)$?

SUBSTITÚCIA TERMOV

Toto rieši *substitúcia termov* za premenné v literáloch. Premenné, ktoré sa vyskytujú vo vete, ktorá je v CNF tvare, sú všeobecne kvantifikované. Náhradou takejto premennej napríklad za konštantu dochádza k znižovaniu počtu uvažovaných interpretácií (namiesto mnohých rozšírených interpretácií bola zvolená iba jedna). To však nevadí, pretože cieľom je realizovať dôkaz sporom – a nepravdivosť v jednej rozšírenej interpretácii znamená celkovú nepravdivosť vďaka univerzálnej kvantifikácii premennej.

UNIFIKÁCIA

Cieľom *unifikácie* je nájsť takú substitúciu, aby dve vety boli ekvivalentné. Potom pod zhodnosťou dvoch predikátov sa rozumie situácia, keď tieto dva predikáty sú navzájom unifikovateľné. Formálne pod unifikáciou dvoch viet P a Q sa rozumie taká operácia

$$unify(P, Q) = \theta$$

ktorej výsledkom je *unifikátor* θ, umožňujúci dosiahnuť rovnakú podobu oboch viet v prípade, že na obe bude aplikovaná substitúcia daná týmto unifikátorom, a teda bude platiť

$$subst(\theta, P) = subst(\theta, Q)$$

UNIFIKÁTOR

Unifikátor je vlastne množina substitúcií premenných, pričom pre každú premennú môže obsahovať najviac jednu náhradu. V prípade ilustračných viet $p(X, f(X))$ a $p(g(Z), Y)$ by unifikátor mohol mať trebárs jeden z tvarov

$$\theta = \{X/g(Z), Y/f(g(Z))\} \qquad \theta = \{X/g(Z), Y/f(g(Z)), Z/q\}$$

kde zápis a/b znamená, že pri substitúcii s použitím daného unifikátora je a nahrádzané pomocou b (teda napríklad v druhom prípade premenná Z bola nahradená konštantou q). Je zrejmé, že úloha nájdenia unifikátora nemusí poskytnúť jednoznačné riešenie – principiálne existuje viacero možných unifikátorov. Tieto unifikátory za navzájom líšia na základe svojej všeobecnosti – všeobecnejší je ten z nich, ktorý menej obmedzuje hodnoty premenných (teda v predchádzajúcej ukážke ľavý unifikátor je všeobecnejší, pretože na

rozdiel od pravého neobmedzuje hodnotu premennej Z). Aby sme sa vyhli nejednoznačnosti, tak sa zvykne požadovať nájdenie *najvšeobecnejšieho* unifikátora.

ZOVŠEOBECNENÁ REZOLVENCIA

V prípade, že pri rezolvencii vystupujú klauzuly obsahujúce aj premenné, je nutné pre dosiahnutie komplementarity dvoch literálov nájsť vhodný unifikátor, ktorý vhodným spôsobom obmedzí premenné v týchto dvoch literáloch. Toto obmedzenie sa musí preniesť aj na výslednú rezolventu (pretože je rezolventou iba za platnosti obmedzenia daného nájdeným unifikátorom). Rezolvenčné odvodzovacie pravidlo bude mať potom o niečo všeobecnejší tvar

$$\frac{P \vee Q \qquad \neg R \vee S}{subst(\theta, P \vee S\ )}$$

kde unifikátor θ je taký, že $subst(\theta, Q) = subst(\theta, R)$.

KOLÍZIA PREMENNÝCH

Špeciálnym prípadom je situácia, keď v oboch vstupných klauzulách sa vyskytuje premenná rovnakého mena. Dôvodom je to, že platí ekvivalentnosť

$$\forall X\ p(X) \wedge g(X)\ \equiv\ \forall X\ \forall Y\ p(X) \wedge g(Y)$$

Keďže premenné, ktoré sa nachádzajú v klauzulách vety v tvare CNF sú implicitne univerzálne kvantifikované, tak potom dôsledkom je, že

- literály $p(X, f(r))$ a $\neg p(g(r), X)$ sú komplementárne,
- rezolventou klauzúl $p(X) \vee q(r)$ a $\neg q(X)$ nie je $p(r)$,
- rezolventou klauzúl $p(X) \vee q(r)$ a $\neg q(r) \vee s(X)$ nie je $p(X) \vee s(X)$,

pretože ak v druhej vstupnej klauzule sa zmení označenie premennej z X na Y, potom pre prvý prípad bude existovať unifikátor $\theta = \{X/g(r), Y/f(r)\}$, v druhom prípade rezolventou bude $p(X)$ a v treťom prípade bude rezolventou klauzula $p(X) \vee s(Y)$.

Preto je ešte pred samotnou rezolvenciou potrebné premenovať premenné tak, aby nenastal prípad výskytu rovnakého mena premennej v oboch vstupných klauzulách. Alternatívne to je možné zabezpečiť ako dodatočný krok č. 8 v algoritme Alg. 2.1 pre transformáciu viet do tvaru CNF – v klauzulách sa premenujú premenné tak, aby v žiadnej z dvojíc klauzúl sa nevyskytovalo rovnaké meno premennej.

PROCES UNIFIKÁCIE

Pre proces unifikácie v zásade platia dve pravidlá, ktoré určujú prípady, keď unifikácia nie je možná. Sú to

- konštanta ani funkcia nemôžu byť nahradené premennou,

Rozšírenie: Redukcia rovnosti

Paramodulácia

Binárny predikát $< \text{term} > = < \text{term} >$ (vyjadrovaný častejšie v infixnom než v tradičnom prefixnom tvare) reprezentujúci rovnosť dvoch termov má oproti ostatným predikátom význačné postavenie. Môže byť síce eliminovaný pomocou rezolvencie ako každý iný predikát, avšak vďaka jeho sémantike sú aj iné spôsoby jeho eliminácie. Reprezentantom týchto iných spôsobov je *paramodulácia* založená na použití *paramodulačného pravidla*

$$\frac{P[t] \qquad (r = s) \vee Q}{subst(\theta, P[s] \vee Q\ )}$$

kde $P[t]$ je klauzula, ktorá obsahuje term t ako svoju súčasť (klauzula je kontextom pre tento term) a samotný term t nie je premennou). Potom θ je unifikátor, ktorý unifikuje term t a term r tak aby platilo

$$subst(\theta, t) = subst(\theta, r)$$

Intuitívne, ak $subst(\theta, Q)$ nie je pravdivá veta, potom musí platiť $subst(\theta, r) = subst(\theta, s)$. A teda $subst(\theta, t)$ môže byť nahradené nielen ekvivalentným $subst(\theta, r)$ ale aj odvodeným $subst(\theta, s)$. Príkladom paramodulácie je

$$\frac{q(X) \vee p(f(X)) \qquad (f(a) = g(a)) \vee q(b)}{q(a) \vee p(g(a)) \vee q(b)}$$

Paramodulácia sa často používa s cieľom nahradiť "väčšie" termy "menšími", teda je ohraničená na prípady, keď $subst(\theta, s)$ nie je podľa nejakého kritéria považované za väčšie ako $subst(\theta, r)$.

- premenná nemôže byť nahradená termom, ktorý obsahuje danú premennú,

a teda unifikátor nemôže obsahovať substitúciu typu p/X resp. $f(p)/X$ ani $X/f(X)$, kde p reprezentuje konštantu, f funkčný symbol a X zase premennú.

Unifikačný algoritmus

Toto je obsiahnuté v unifikačnom algoritme Alg. 2.2. Algoritmus začína s prázdnym unifikátorom, pričom postupne unifikované výrazy rozkladá na jednoduchšie prvky a tie sa pokúša navzájom unifikovať. V prípade, že unifikácia na nejakom mieste skončí neúspechom (reprezentovaným pomocou symbolu $\square$), tak aj celkový pokus o unifikáciu je neúspešný a nemá zmy-

vstup: predikáty x a y
výstup: unifikátor θ v prípade úspechu alebo $\square$ pri neúspechu

```
    unify(x, y)
1.      θ := {}
2.      return unify_aux(x, y, θ)

    unify_aux(x, y, θ)
1.      if θ = □ then return □
2.      else if x = y then return θ
3.      else if var?(x) then return unify_var(x, y, θ)
4.      else if var?(y) then return unify_var(y, x, θ)
5.      else if x = a(p, ..., r) and y = b(s, ..., u) then return
6.              unify_aux([p, ..., r], [s, ..., u], unify_aux(a, b, θ))
7.      else if x = [p, q, ..., r] and y = [s, t, ..., u] then return
8.              unify_aux([q, ..., r], [t, ..., u], unify_aux(p, s, θ))
9.      else return □

    unify_var(var, x, θ)
1.      if var/y ∈ θ then return unify_aux(y, x, θ)
2.      else if x/y ∈ θ then return unify_aux(var, y, θ)
3.      else if var ∈ x then return □
4.      else return θ = θ ∪ {var/x}
```

Alg. 2.2: Unifikačný algoritmus

sel si uchovať dovtedy vybudovaný unifikátor. Pokiaľ unifikované objekty sú zložené (teda sú predikátmi alebo funkciami), tak sa najprv unifikujú ich symboly a nato zoznamy ich argumentov. Ak unifikované objekty sú zoznamami (reprezentujúcimi zoznamy argumentov funkcií alebo predikátov), tak sa unifikácia vykonáva rekurzívne – najprv sú unifikované prvé prvky zoznamov a následne sa unifikujú zvyšky týchto zoznamov bez prvých prvkov. Ak niektorý unifikovaný prvok je premennou (pre testovanie sa používa *var*?), tak algoritmus prejde na špeciálnu sekciu pre unifikáciu

premennej. A ak ani jedna z uvedených možností nie je vhodná, znamená to že dané prvky sa nedajú unifikovať a signalizuje sa neúspech.

UNIFIKÁCIA PREMENNEJ

V časti pre unifikáciu premennej sa pred vytvorením substitúcie vykonáva niekoľko kontrol. Najprv sa kontroluje, či sa daná premenná už skôr nevyskytla v nejakej substitúcii – ak áno, tak sa tá substitúcia použije (pretože nie je možné pre premennú vytvoriť viac substitúcií). Keďže druhým argumentom pre $unify\_var$ môže byť tiež premenná, tak sa toto skúmanie robí aj pre tento argument. A záverečne sa kontroluje, či sa premenná (prvý argument) nevyskytuje ako súčasť druhého argumentu (to by došlo k porušeniu jedného z uvedených pravidiel pre unifikáciu). Ak všetky tieto kontroly sú v poriadku, tak sa vytvorí nová substitúcia a zaradí sa do unifikátora.

ALGORITMUS PRE DÔKAZ SPOROM

Po vysvetlení všetkých potrebných prvkov je možné prezentovať formálnu podobu algoritmu pre dôkaz sporom, ktorá je uvedená ako Alg. 2.3. Jedná sa o cyklický proces, kde v každej iterácii sa z množiny uchovávaných klauzúl K vyberajú dve klauzuly, vytvára sa z nich množina rezolvent (všetky rezolventy, ktoré je možné z dvoch vybraných klauzúl vytvoriť) a tieto rezolventy sa následne môžu pridať k uchovávaným klauzulám.

---

vstup: znalostná báza KB a hypotéza H v klauzulárnom tvare
výstup: $TRUE$ v prípade dokázania alebo $FALSE$ inak

---

1. $K := KB \cup \neg H$
2. $K := K \setminus \{$ *D: D obsahuje čistý literál* $\}$
3. **while** $\neg(\square \in K)$ **do**
4. *vyber* $C_1, C_2 \in K$
5. **if** *žiadny nový pár* C_1, C_2 *neexistuje* **then return** $FALSE$
6. $R := \{D : D$ *je rezolventa* C_1 a C_2, *nie je tautológia* $\}$
7. **for** $D \in R$ do
8. **if** *žiadna klauzula z* K *nezahŕňa* D
9. **then** $K := \{D\} \cup \{C \in K : D$ *nezahŕňa* $C\}$
10. **return** $TRUE$

Alg. 2.3: Rezolvenčný dôkaz sporom

Rozšírenie: Zodpovedanie otázok

Výber informácií

Predpokladajme, že v ilustračnom príklade Pr. 2.1 agent už pochodil časť priestoru a vykonal niektoré odvodenia (napr. že priepasť nie je na pozíciách $pole(1,3)$ a $pole(2,2)$ a že strašidlo sa nenachádza na pozíciách $pole(3,1)$ a $pole(2,2)$). Agent práve stojí na pozícii $pole(1,2)$ a potrebuje zistiť na základe svojej znalostnej bázy, ktoré pozície v jeho okolí sú bezpečné. Otázka na vyhľadanie bezpečných pozícií by mohla mať tvar

$$\forall X \; susedne(pole(1,2), X) \wedge \neg priepast(X) \wedge \neg strasidlo(X)$$

pričom táto otázka bude hrať úlohu hypotézy. Transformácia tejto hypotézy na tvar CNF a jej negácia poskytne cieľovú klauzulu

$$\neg susedne(pole(1,2), X) \vee priepast(X) \vee strasidlo(X)$$

ktorú je možné pomocou série rezolvencií redukovať na prázdnu klauzulu použitím $\neg po(A,C) \vee susedne(pole(A,B), pole(C,B))$, $po(1,2)$, $\neg priepast(pole(2,2))$ a $\neg strasidlo(pole(2,2)$. Problémom je však to, že algoritmus Alg. 2.3 vráti $TRUE$ namiesto presného určenia bezpečnej pozície – teda podarilo sa dokázať, že bezpečná pozícia v okolí agenta existuje, ale agent sa nedozvedel, ktorá to je.

V prípade, že cieľom nie je iba potvrdenie alebo vyvrátenie nejakej hypotézy ale výber informácií zo znalostnej bázy, možno k otázke doplniť ďalší "informačný" predikát, ktorý sa nenachádza v žiadnej klauzule v znalostnej báze. Argumenty tohto predikátu budú reprezentovať hľadanú informáciu. V našom prípade by potom po doplnení cieľová klauzula mala tvar

$$\neg susedne(pole(1,2), X) \vee priepast(X) \vee strasidlo(X) \vee info(X)$$

Keďže pre tento doplnený predikát v znalostnej báze neexistuje žiadny komplementárny predikát, nie je možná redukcia na prázdnu klauzulu, pretože nie je možnosť ho pomocou rezolvencie z klauzuly odstrániť.

Z tohto dôvodu algoritmus Alg. 2.3 musí byť upravený tak, aby namiesto testu na výskyt prázdnej klauzuly hľadal takú klauzulu, ktorá obsahuje iba doplnený informačný predikát (riadok 3 algoritmu) a v prípade úspechu namiesto $TRUE$ vrátil argumenty daného informačného predikátu (riadok 10). Aby to bolo možné, doplnený informačný predikát musí mať vopred definované meno, ktoré je algoritmu známe.

Pri vytváraní rezolvent sa uvažujú iba tie rezolventy, ktoré nie sú tautológiami (rezolventa je tautológiou, ak súčasne obsahuje nejaký literál v priamej aj negovanej podobe), pretože v prípade tautológie by mohlo nastať

Vylúčenie tautológií

$$\dfrac{\dfrac{\dfrac{\dfrac{\neg P \vee Q \vee R \quad \neg Q \vee P}{\neg P \vee R \vee P} \quad \neg R}{\neg P \vee P} \quad \neg P}{\neg P}}{}$$

kde je vidno, že tautológia v konečnom dôsledku neumožní získať prázdnu klauzulu (teda dôkaz sporom nemôže obsahovať žiadnu tautológiu) a preto je potrebné sa takýchto rezolvent zbaviť čo najskôr.

Pri pridávaní rezolvent k ostatným klauzulám ešte hrá úlohu vlastnosť *zahrnutia*. Klauzula C zahŕňa klauzulu D vtedy, ak existuje taký unifikátor θ, že platí $subst(\theta, C) \subseteq D$ (C je zahŕňajúcou a D zase zahrnutou klauzulou), teda klauzula D obsahuje tie isté literály ako klauzula $subst(\theta, C)$ (okrem nich môže samozrejme obsahovať aj ďalšie literály). Tak napríklad klauzula $q(X)$ zahŕňa klauzulu $\neg p(a) \vee q(a)$ kvôli existencii $\theta = \{X/a\}$. Síce podobne aj klauzula $q(X) \vee q(Y)$ zahŕňa klauzulu $q(a)$, avšak v algoritme sú zvyčajne uvažované iba tie prípady zahŕňania, kedy zahŕňajúca klauzula nie je dlhšia ako klauzula zahrnutá. V prípade súčasného výskytu zahŕňajúcej a aj zahrnutej klauzuly nie je potrebné aj naďalej uvažovať zahrnutú klauzulu, pretože tým istým postupom, ktorý redukuje zahrnutú klauzulu na prázdnu, je možné redukovať aj zahŕňajúcu klauzulu na prázdnu klauzulu a to tým skôr, čím je zahŕňajúca klauzula kratšia. Toto možno ilustrovať na príklade

Vylúčenie zahrnutých klauzúl

$$\dfrac{\dfrac{\dfrac{\neg P \vee Q \vee R \quad \neg Q}{\neg P \vee R} \quad \neg R}{\neg P} \quad P}{\square} \qquad \dfrac{\dfrac{Q \vee R \quad \neg Q}{R} \quad \neg R}{\square}$$

kde $Q \vee R$ je zahŕňajúcou a $\neg P \vee Q \vee R$ zahrnutou klauzulou. Na základe zahrnutia je možné, že niektorá rezolventa nie je pridaná k ostatným klauzulám alebo naopak, pridanie rezolventy spôsobí vypustenie nejakej už skôr pridanej klauzuly.

Navyše k uvedeným dvom špeciálnym prípadom (tautológie a zahrnutia) je možné vypustiť aj klauzuly, ktoré obsahujú taký predikát, že žiadna z klauzúl neobsahuje daný predikát v negovanom tvare (riadok 2 algoritmu) – v tomto prípade nie je možné uskutočniť žiadnu takú rezolvenciu, ktorá by daný predikát odstránila z výslednej rezolventy.

Vylúčenie klauzúl s čistým literálom

Ukončenie algoritmu môže nastať z dvoch dôvodov – alebo bola vygenerovaná prázdna klauzula (a dôkaz bol úspešný) alebo prázdna klauzula

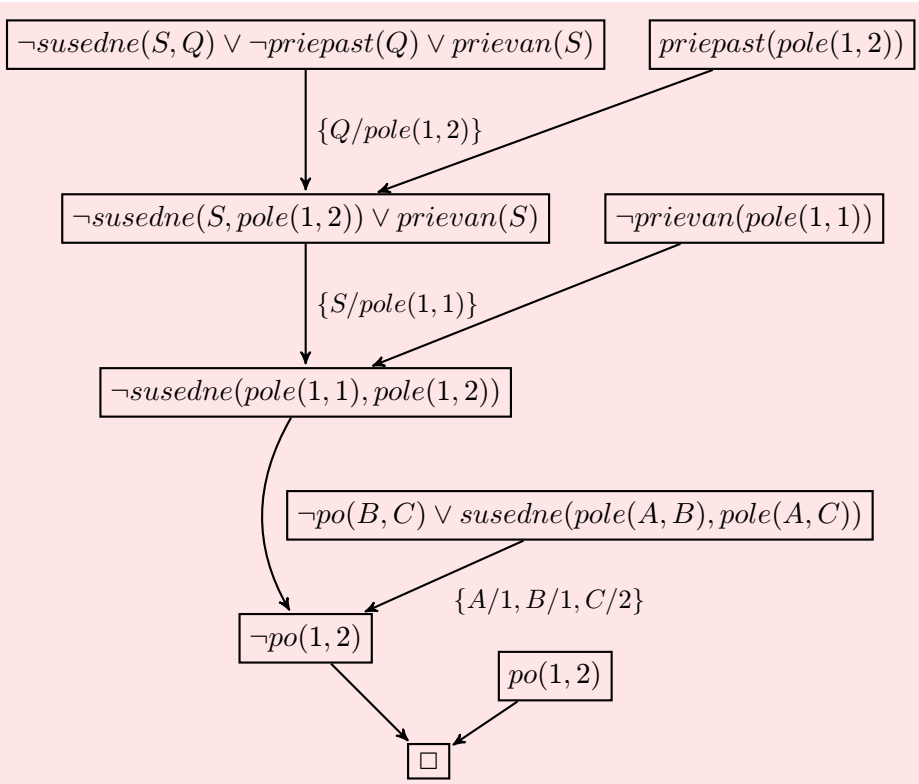
nebola vygenerovaná a zároveň už nie je možné vytvoriť takú rezolventu, ktorá ešte vytvorená nebola.

Ako ukážku rezolvenčného uvažovania opäť zoberme ilustračný príklad Pr. 2.1 s tým, že agent sa nachádza na pozícii $pole(1,1)$ a na tejto pozícii nevníma ani prievan ani zápach. Skúša zistiť, či pozícia $pole(1,2)$ je bezpečná, teda či sa na danej pozícii dá vylúčiť prítomnosť strašidla aj priepasti. Pre vylúčenie priepasti si stanoví hypotézu $\neg priepast(pole(1,2))$, ktorú sa pokúsi dokázať. Hypotézu teda neguje a snaží sa v dôkaze získať prázdnu klauzulu.

DERIVAČNÝ GRAF

Derivačný graf je zobrazený na Obr. 2.1, pričom sú zobrazené iba tie rezolvencie, ktoré prispievajú k redukcii na prázdnu klauzulu (a teda vďaka tomu má graf stromovú štruktúru). Uzly grafu reprezentujú klauzuly dvojakého typu:

- vstupné klauzuly (klauzuly vybraté zo znalostnej bázy),



Obr. 2.1: Derivačný graf

- rezolventy.

Prvý typ klauzúl je znázornený uzlami, ktoré nemajú predchodcov, zatiaľ čo uzly s predchodcami reprezentujú druhý typ klauzúl. Hrany reprezentujú uskutočnené rezolvencie, keď vždy dve hrany spájajú vstupné klauzuly s ich rezolventou. V prípade, že bolo potrebné nájsť pre nejaký rezolvenčný krok unifikátor, tak tento je v grafe znázornený tiež.

REDUNDANTNÉ REZOLVENCIE

Okrem uvedených rezolvencií však mohli byť vykonané aj ďalšie. Takými mohli byť napríklad rezolvencia klauzuly $\neg susedne(S,Q) \vee \neg priepast(Q) \vee prievan(S)$ s klauzulou $\neg po(B,C) \vee susedne(pole(A,B), pole(A,C))$ či rezolvencia prvej klauzuly s klauzulou $\neg prievan(pole(1,1))$. Obe tieto rezolvencie by vytvorili alternatívne cesty v derivačnom grafe k redukcii na prázdnu klauzulu.

ZBYTOČNÉ REZOLVENCIE

Naproti tomu rezolvencia klauzuly $\neg susedne(S,Q) \vee \neg priepast(Q) \vee prievan(S)$ s klauzulou $\neg po(C,B) \vee susedne(pole(A,B), pole(A,C))$ vedie do slepej uličky, pretože pri aktuálnych znalostiach neumožní redukciu na prázdnu klauzulu (ostala by rezolventa $\neg po(2,1)$, ktorú už nejde ďalej redukovať). Je teda zrejmé, že výber dvojíc klauzúl, ktoré majú byť rezolvované, má veľký vplyv na to, koľko rezolvent bude generovaných dovtedy, kým sa podarí odvodiť prázdnu klauzulu (a najmä koľko rezolvent bude generovaných zbytočne, pretože neprispejú k odvodeniu prázdnej klauzuly).

Je možné používať rozličné stratégie voľby klauzúl, ktoré by mali byť navzájom rezolvované (riadok 4 algoritmu Alg. 2.3), s cieľom minimalizovať počet rezolvencií. Najznámejšími stratégiami sú

- usporiadanie podľa veľkosti,
- jednotková preferencia,
- oporná množina,
- hĺbková saturácia.

STRATÉGIA USPORIADANIA PODĽA VEĽKOSTI

Pri *usporiadaní podľa veľkosti* sa preferujú malé klauzuly pred klauzulami väčšími, pričom ich veľkosť sa určuje podľa predpisu:

- $\| X \| = 1$ (pre premennú),
- $\| a \| = 1$ (pre konštantu),
- $\| f(t_1, ..., t_n) \| = 1 + \| t_1 \| + ... + \| t_n \|$ (pre funkcie a predikáty),
- $\| \neg L \| = \| L \|$

- a nakoniec veľkosť klauzúl sa určí podľa $\| L_1 \vee ... \vee L_n \| = \| L_1 \| + ... + \| L_n \|$.

Preferencia malých klauzúl je založená na snahe, aby produkované rezolventy boli čo najmenšie (a tým pádom čo najrýchlejšie redukovateľné na prázdnu klauzulu).

STRATÉGIA JEDNOTKOVEJ PREFERENCIE

Pri *jednotkovej preferencii* sa používajú jednotkové klauzuly s jediným literálom. Stratégia je založená na fakte, že v prípade že jedna z klauzúl obsahuje iba jeden literál, je rezolventa kratšia než druhá použitá klauzula. Použitie tejto stratégie je založené na nádeji, že explicitný dôraz na skracovanie klauzúl umožní skôr dosiahnuť prázdnu klauzulu. Kvôli úplnosti stratégia dovoľuje aj rezolvenciu dlhších klauzúl, avšak jednotkové klauzuly sú preferované a vyberané pred dlhšími.

STRATÉGIA OPORNEJ MNOŽINY

Pri stratégii *opornej množiny* sa vychádza z myšlienky, že samotná znalostná báza KB je neprotirečivá a teda nie je možné odvodiť prázdnu klauzulu iba na základe klauzúl znalostnej bázy. Protirečivosť je spôsobená až pridaním negácie dokazovanej hypotézy. Preto v každej rezolvencii by ako jedna z klauzúl mala byť vybratá buď priamo táto negovaná hypotéza alebo nejaká klauzula, ktoré bola z tejto negovanej hypotézy odvodená (negovaná hypotéza a všetky rezolventy odvodené z nej priamo alebo nepriamo vytvárajú opornú množinu).

STRATÉGIA HĹBKOVEJ SATURÁCIE

Pri *hĺbkovej saturácii* všetky klauzuly vo vstupnej množine sú umiestnené v hĺbke 0. Rezolventa je v hĺbke $n + 1$ vtedy, ak jedna jej rodičovská klauzula je v hĺbke n a druhá v hĺbke menšej alebo rovnej n. Stratégia predpisuje najprv odvodiť všetky možné rezolventy v hĺbke 1, potom všetky v hĺbke 2, atď. Je to systematická stratégia – je vlastne analógiou *prehľadávania do šírky*.

Cvičenia

1. Reprezentujte nasledovné vyjadrenia v prirodzenom jazyku pomocou predikátového počtu prvého rádu a transformujte ich do tvaru CNF:

   - V meste je holič, ktorý holí všetkých mužov v meste, ktorí sa neholia sami.
   - Neexistuje také miesto, kde každý na tom mieste je chytrý vtedy a len vtedy, ak každý je na tom mieste a je chytrý.
   - Všade je niekto, kto je tam a je chytrý ako aj je niekto, kto je chytrý ak tam je.
   - Bratranci sú deti súrodencov.

Rozšírenie: Vyjadrenie zmien

Situačný počet

Predpokladajme, že v ilustračnom príklade Pr. 2.1 je potrebné reprezentovať aj pohyb agenta. Napríklad jeho úlohou je nájsť pole s uloženým pokladom a ten zdvihnúť, pričom sa môže pohybovať vždy iba o jeden krok na susedné pole, aj to iba vtedy ak to pole je bezpečné. V tomto prípade by mala byť súčasťou reprezentácie aj aktuálna poloha agenta. A tu nastáva problém – v logike nemôže niečo chvíľu platiť a potom chvíľu neplatiť (napríklad agent je na začiatku na pozícii $pole(1,1)$ ale po svojom odchode z tejto pozície tam už nie je).

Vyjadrenie zmien pri použití predikátovej logiky umožňuje *situačný počet*. V ňom sa zmeny stavu sveta reprezentujú pomocou situácií, pričom situácia reprezentuje nejaký časový úsek, v ktorom je stav nemenný. To, že platnosť predikátu $pozicia(pole(1,1))$ reprezentujúceho polohu agenta je viazaná na čas, sa vyjadrí doplnením dodatočného "situačného" argumentu. Jeho platnosť v situácii s_1 bude reprezentovaná pomocou $pozicia(pole(1,1), s_1)$ a jeho neplatnosť v situácii s_2 zase pomocou $\neg pozicia(pole(1,1), s_2)$.

Ak došlo k nejakej zmene, tak je potrebné uvažovať vždy dve situácie – situáciu pred zmenou a situáciu po zmene. Zmena stavu je potom opísaná zmenou jednej situácie na druhú ako následok nejakej akcie. Ak je potrebné zachovať informáciu, ktorá akcia spôsobila daný prechod, tak situáciu možno reprezentovať ako funkciu predchádzajúcej situácie a danej akcie. Pri takejto reprezentácii by sa namiesto $\neg pozicia(pole(1,1), s_2)$ použilo $\neg pozicia(pole(1,1), f(krok, s_1))$. Teda iba prvá (počiatočná) situácia je pomenovaná konštantou, všetky nasledujúce situácie sa rekurzívne odvádzajú od nej. Inou možnosťou by bolo namiesto funkcie f použiť funkciu $krok$. Fakt o neplatnosti pozície by sa potom dal vyjadriť $\neg pozicia(pole(1,1), krok(s_1))$.

Pri použití situačného počtu je možné v reprezentácii úlohy rozoznať niekoľko rôznych typov viet:

- fakty popisujúce počiatočný stav – všetky predikáty viažúce sa k počiatočnému stavu. Príkladom je $pozicia(pole(1,1), s_0)$ reprezentujúca štartovaciu polohu agenta.
- všeobecné tvrdenia – platné v každej situácii. Príkladom je vzťah medzi jasom na danej pozícii v nejakej situácii a výskytom pokladu na tej istej pozícii v tej istej situácii (prítomnosť pokladu je indikovaná jasným svetlom)

$$\forall P, S \; jas(P, S) \leftrightarrow poklad(P, S)$$

Rozšírenie: Vyjadrenie zmien (pokr.)

alebo vzťah medzi prievanom/zápachom na nejakej pozícii a priepasťou/strašidlom na susednej pozícii. Vety tohto typu sú vyjadrené buď iba s použitím jednej situácie (ak obsahujú predikáty, ktoré sú situačne závislé) ako v ukážke alebo bez uvažovania situácie, napríklad definícia susednosti polí, teda bez pridávania situačného argumentu.

- účinky akcií – vyjadrenie zmien (zmeny platnosti stavových predikátov a prechod na nasledujúcu situáciu). Príkladom je zmena pozície pri vykonaní kroku na susedné bezpečné pole

$$\forall P, R, S \quad pozicia(P, S) \wedge susedne(P, R) \wedge \neg priepast(R) \wedge \neg strasidlo(R) \rightarrow pozicia(R, f(krok, S))$$

alebo privlastnenie si pokladu

$$\forall P, S \quad pozicia(P, S) \wedge poklad(P, S) \rightarrow \neg poklad(P, f(zober, S)) \wedge bohaty(f(zober, S))$$

- rámcové axiómy – vyjadrenie, že nejaký fakt popisujúci stav sa nezmení napriek prechodu k novej situácii. Príkladom je nemennosť polohy pokladu pri pohybe agenta alebo nemennosť polohy agenta pri privlastňovaní si pokladu

$$\forall P, S \; poklad(P, S) \rightarrow poklad(P, f(krok, S))$$

$$\forall P, S \; pozicia(P, S) \rightarrow pozicia(P, f(zober, S))$$

Cieľ, ktorý sa má dosiahnuť (charakterizujúci konečný stav) je hypotézou, ktorú je potrebné dokázať. Keďže hľadanou informáciou je nielen to, či sa daný cieľ dá dosiahnuť, ale aj postupnosť transformácií, ktorá ho umožnila dosiahnuť z počiatočného stavu, tak cieľovou hypotézou by bol výraz $bohaty(S)$ s dodatkom pre zodpovedanie otázky na hodnotu situačnej premennej S.

- Všetci čitatelia si v každej knihe, ktorú čítali, našli svoju obľúbenú postavu práve vtedy, keď si v nej našli aj neobľúbenú postavu.

2. Unifikujte nasledovné dvojice výrazov

   - $p(f(X,a),g(Y,Y),Z)$ $\quad$ $p(f(g(a,b),Z),X,a)$
   - $p(X,X,Z)$ $\quad$ $p(f(a,a),Y,Y)$
   - $p(X,f(Y,Z),b)$ $\quad$ $p(g(a,Y),f(Z,g(a,X)),b)$
   - $p(a,Y,U)$ $\quad$ $p(X,f(X,U),g(Z,b))$

3. Pre ilustračný príklad Pr. 2.1 prepíšte znalosti o príklade do tvaru CNF. Simulujte pohyb agenta, jeho vnemy postupne pridávajte do znalostnej bázy a na základe nich odvádzajte nové znalosti o jednotlivých poliach.

4. Vyriešte nasledujúcu detektívnu záhadu jedného tragického konca

   - Každého, kto má rád všetky zvieratá, má niekto rád.
   - Kohokoľvek, kto zabil nejaké zviera, nemá nikto rád.
   - Jack má rád všetky zvieratá.
   - Alebo Jack alebo zvedavosť zabili mačku, ktorá sa volala Fúzik.

   Bola vinníkom zvedavosť ?

5. Pri úlohe farbenia grafov je zadaný graf, ktorého uzly je potrebné zafarbiť – každému vrcholu priradiť jednu farbu tak, aby žiadne dva vrcholy, ktoré sú spojené hranou, neboli zafarbené rovnakou farbou. Počet farieb je vopred daný.

   Dokážte, že pre zafarbenie grafu s piatimi uzlami stačia štyri farby v prípade, že graf nie je plne prepojený, avšak v prípade, že každý uzol je spojený s každým iným uzlom, štyri farby nestačia.

6. Reprezentácie predchádzajúcich príkladov prepíšte do požadovaného tvaru a ich výsledky overte použitím implementácie algoritmu rezolvenčného dokazovania.

7. Šesť umeleckých diel (C, D, E, F, G a H) má byť vystavených v troch miestnostiach (1, 2 a 3) galérie. Výstava musí splniť tieto podmienky:

   - Diela C a E by nemali byť vystavené v rovnakej miestnosti.
   - Diela D a G musia byť vystavené v rovnakej miestnosti.
   - Ak E a F sú vystavené v rovnakej miestnosti, tak v tej miestnosti už by nemalo byť žiadne ďalšie dielo.
   - Aspoň jedno dielo musí byť vystavené v každej z miestností, najviac však tri diela v jednej miestnosti.

Ak dielo D je vystavené v miestnosti 3 a diela E a F sú vystavené v miestnosti 1, ktoré z nasledujúcich tvrdení sú pravdivé ?

- Dielo C je vystavené v miestnosti 1 (iné alternatívy: G v miestnosti 2, H v 1).
- Nie viac ako dve diela sú vystavené v miestnosti 3.
- Diela F a H sú vystavené v rovnakej miestnosti (iná alternatíva: C a H sú spolu).
- Tri diela sú vystavené v miestnosti 2.

8. Uvažujte hádanku "Bratov ani sestier nemám, ale otec toho muža je synom môjho otca". Napíšte pravidlá pre doménu rodinných vzťahov a použite ich pre zistenie, kto je tým mužom.

9. Predpokladajme, že na základe znalostnej bázy $\{ \forall X\ c(X) \rightarrow g(X), \forall X\ a(X) \rightarrow \neg g(X), \exists X\ a(X) \wedge b(X) \}$ je potrebné dokázať hypotézu $\exists X\ b(X) \wedge \neg c(X) \wedge \neg g(X)$. Dokážte ju, pričom pri rezolvencii použijete pre výber klauzúl stratégiu:

   - hĺbkovej saturácie,
   - usporiadania podľa veľkosti,
   - jednotkovej preferencie,
   - opornej množiny,
   - lexikografického (abecedného) usporiadania klauzúl.

   Porovnajte stratégie z hľadiska počtu generovaných rezolvent a veľkostí vytvorených derivačných grafov.

10. Napíšte pravidlá pre svet troch kociek, ktoré je možné ukladať jednu na druhú či odkladať na podložku. Využite ich pre vytvorenie plánu, pomocou ktorého je možné transformovať počiatočnú konfiguráciu na cieľovú, pričom vyskúšajte rôzne počiatočné a cieľové kombinácie tak, aby potrebný plán pozostával z

    - jednej akcie,
    - dvoch akcií,
    - troch akcií,
    - štyroch akcií.

Kapitola 3

Hornova logika

Jednou veľmi zaujímavou podtriedou klauzúl (či už výrokového počtu alebo predikátového počtu prvého rádu) sú tzv. *Hornove klauzuly*. Sú to také klauzuly, ktoré obsahujú najviac jeden pozitívny literál (a teda sú rozšírením *definitných* klauzúl, ktoré obsahujú práve jeden pozitívny literál). Klauzula môže okrem pozitívneho obsahovať aj niekoľko negatívnych literálov. Napríklad tvrdenie, že luhár neustále klame, sa dá vyjadriť ako

HORNOVE KLAUZULY

$$\neg luhar(Kto) \vee \neg povedal(Kto, Co) \vee loz(Co)$$

kde prvé dva literály sú negatívne a posledný je pozitívny. Hornova klauzula takéhoto tvaru vlastne reprezentuje implikáciu

$$luhar(Kto) \wedge povedal(Kto, Co) \rightarrow loz(Co)$$

označovanú ako *pravidlo*, pričom negatívne literály reprezentujú predpoklady pravidla (označované tiež ako telo pravidla alebo LHS – ľavá strana pravidla) a pozitívny literál reprezentuje záver (označovaný aj ako hlava pravidla alebo RHS – pravá strana pravidla).

PRAVIDLO

Príkladom iného typu klauzuly je $povedal(jano, \, 'dnes \; svieti \; slnko')$, čo je skrátený zápis formálneho pravidla

$$\top \rightarrow povedal(jano, \, 'dnes \; svieti \; slnko')$$

kde $\top$ je predikátový symbol, ktorý je interpretovaný ako vždy pravdivý. Takáto klauzula bez negatívnych literálov sa označuje ako *fakt*.

FAKT

A nakoniec je možná aj taká klauzula, ktorá neobsahuje žiadny pozitívny literál, iba negatívne literály. Príkladom takéhoto typu klauzuly je $\neg loz('dnes \; svieti \; slnko') \vee \neg luhar(jano)$, čo je skrátený zápis formálneho pravidla

$$loz('dnes \; svieti \; slnko') \wedge luhar(jano) \rightarrow \bot$$

ILUSTRAČNÝ PRÍKLAD

Predpokladajme existenciu ostrova, na ktorom žije komunita ľudí, na ktorých navonok nevidieť nič zvláštne až do doby, kým sa s nimi niekto nepokúša komunikovať. Pravdou je, že obyvatelia sú rozdelení do dvoch kmeňov – kmeňa poctivcov a kmeňa luhárov. Ich príslušnosť nie je navonok nijako zrejmá okrem ich slov:

- *príslušníci kmeňa poctivcov hovoria vždy pravdu,*
- *príslušníci kmeňa luhárov za každých okolností klamú.*

Úloha 1: *Keďže sa obyvatelia navzájom poznajú, každý z nich je schopný o každom povedať, aká je jeho príslušnosť k niektorému z kmeňov. Predpokladajme, že na ostrove okrem iných žijú Jano a Juro (obaja pochádzajú z kmeňa poctivcov) ako aj Fero (z kmeňa luhárov). Čo povie každý z nich o každom z ostrovanov?*
Úloha 2: *Aby cestovateľ mohol využívať informácie získané od miestnych obyvateľov, musí byť schopný na základe rozhovorov s obyvateľmi určiť, kto z nich je poctivcom a kto luhárom. Príkladom takéhoto rozhovoru je*

- *A povedal: "B je luhár."*
- *B povedal: "A je rovnaký ako C."*

Reprezentácia: *Uvedený problém je reprezentovaný pomocou faktov:*

$$\begin{aligned} pravda(je(poctivec, poctivec)) \quad & ostrovan(jano, poctivec) \\ pravda(je(luhar, luhar)) \quad & ostrovan(juro, poctivec) \\ klamstvo(je(poctivec, luhar)) \quad & ostrovan(fero, luhar) \\ klamstvo(je(luhar, poctivec)) & \end{aligned}$$

pomocou pravidiel:

$$\begin{aligned} ostrovan(Meno, poctivec) \wedge ostrovan(Kto, Kmen) \wedge pravda(je(Kmen, Co)) \\ \rightarrow povedal(Meno, poctivec, je(Kto, Kmen, Co)) \\ ostrovan(Meno, luhar) \wedge ostrovan(Kto, Kmen) \wedge klamstvo(je(Kmen, Co)) \\ \rightarrow povedal(Meno, luhar, je(Kto, Kmen, Co)) \end{aligned}$$

a pomocou cieľa (v prípade úlohy 2):

$$\neg povedal(Am, A, je(Bm, B, luhar)) \vee \neg povedal(Bm, B, je(Am, A, C))$$

Riešenie 1: *Jano označuje seba ako aj Jura za poctivca a Fera za luhára, Juro tvrdí to isté čo Jano. Naproti tomu Fero označuje seba za poctivca a Jana s Jurom za luhárov.*
Riešenie 2: *Do úvahy prichádzajú dve riešenia, keď A a B sú buď poctivcom a luhárom alebo luhárom a poctivcom, pričom C je vždy luhárom.*

Pr. 3.1: Príklad pre reprezentáciu pomocou pravidiel

kde $\bot$ je predikátový symbol vždy interpretovaný ako nepravdivý. Klauzula takéhoto tvaru sa označuje ako *cieľ*. CIEĽ

Pri použití predikátového počtu literály môžu obsahovať premenné – tieto premenné sú považované za univerzálne kvantifikované (avšak samotné kvantifikátory sa vynechávajú). Ak znalostná báza nepoužíva funkčné symboly, je inštanciou *Datalogovej* bázy[1].

Nevýhodou pravidlovej reprezentácie je však to, že nie všetky klauzuly môžu byť transformované do tvaru Hornových klauzúl – našťastie pravidlá sú použiteľné pre širokú škálu úloh (napr. pre úlohy typu "čo sa muselo stať" alebo "čo sa stane" keď sú splnené nejaké podmienky). Na druhej strane výhodou uvedených reštrikcií na tvar klauzúl je to, že nie je nutné pre inferenciu používať plnú silu rezolvencie, ale je možné využiť obmedzenejšie (t.j. menej všeobecné) avšak tým pádom výkonnejšie inferenčné algoritmy. V úlohe týchto algoritmov sa používa algoritmus *dopredného reťazenia* (je základom pre *produkčné systémy*) a algoritmus *spätného reťazenia* (tvorí základ pre *logické programovanie*).

3.1 Dopredné reťazenie pravidiel

Algoritmus dopredného reťazenia používa pravidlá dopredným spôsobom – postupuje od predpokladov k záverom. Jeho podstatou je využitie tých predikátov, o ktorých je známe že platia (a teda v znalostnej báze sú reprezentované faktami). Platné predikáty sa použijú pre odvodenie platnosti ďalších predikátov prostredníctvom tých pravidiel, ktorých predpoklady sú založené na predikátoch s už známou platnosťou. Novo odvodené predikáty môžu umožniť použitie ďalších pravidiel a tým odvodenie platnosti ďalších predikátov – nastáva reťazenie pravidiel v tom zmysle, že použitie jedného pravidla vytvára podmienky pre použitie iného pravidla. Tento proces reťazenia pravidiel rekurzívne pokračuje dovtedy, kým je možné odvodiť nové fakty alebo kým nie je možné rozhodnúť o platnosti nejakého cieľového predikátu. DOPREDNÉ POUŽITIE PRAVIDIEL

Vzhľadom na špecifiká dopredného reťazenia je možné uvažovať špeciálny prípad rezolvenčného pravidla, kde do rezolvencie vstupujú implikácie, z ktorých iba jedna má LHS rôznu od $\top$ (teda iba jedna nereprezentuje platný fakt). REZOLVENČNÉ PRAVIDLO

$$\frac{P_1 \wedge \ldots \wedge P_n \rightarrow S \qquad \top \rightarrow P_1, \ldots, \top \rightarrow P_n}{subst(\theta, S)}$$

[1] Datalog je jazyk, ktorý je obmedzený na definitné klauzuly predikátového počtu prvého rádu bez funkčných symbolov. Často sa využíva ako dopytovací jazyk pre deduktívne databázy alebo pre extrakciu a integráciu dát.

Rezolvencia klauzuly, ktorej LHS je tvorená určitým počtom literálov, s faktom (jednotkovou klauzulou) vyprodukuje skrátenú klauzulu. Takýmto spôsobom je možné z $P_1 \wedge \ldots \wedge P_n \rightarrow S$ pomocou faktu P_n získať klauzulu $P_1 \wedge \ldots \wedge P_{n-1} \rightarrow S$. Po rezolvencii tejto novej klauzuly s faktom P_{n-1} sa získa pravidlo iba s $n-2$ predpokladmi. Opakovanými rezolvenciami s faktami P_{n-3} až P_1 je možné pôvodné pravidlo redukovať na tvar $\top \rightarrow S$, ktorý reprezentuje nový platný fakt.

HYPERREZOLVENCIA

Takáto sekvencia rezolvencií sa vykoná iba vtedy, ak je ju možné doviesť do úspešného konca – ak pôvodnú klauzulu s negatívnymi literálmi je možné pretransformovať na klauzulu bez negatívnych literálov, teda ak je možné pôvodné pravidlo pretransformovať na fakt (so zohľadnením unifikátorov, vytváraných počas sekvencie rezolvencií). Ak sa proces nedá doviesť do konca ale je možné eliminovať iba niektoré z negatívnych literálov, potom sa príslušné rezolvencie nevykonajú. Takýto postup sa označuje ako *hyperrezolvencia* – celá sekvencia rezolvencií sa vykonáva ako jeden krok, keď sa buď vykonajú všetky rezolvencie z danej sekvencie alebo ani jedna. Nie sú produkované žiadne medzivýsledky, čím sa znižujú nároky na pamäťovú kapacitu.

Pre ilustráciu uvažujme existenciu znalostnej bázy, ktorá obsahuje týchto šesť pravidiel

$$\begin{array}{lll} A \wedge B \rightarrow E & B \wedge C \rightarrow E & D \wedge F \rightarrow I \\ D \wedge E \rightarrow G & A \wedge B \wedge C \rightarrow I & E \wedge F \rightarrow H \end{array}$$

ako aj štyri fakty A, B, C a F. Cieľom je dokázať platnosť H. Cieľ je analogický k cieľu dokázať, že báza znalostí po pridaní pravidla $H \rightarrow \bot$ je protirečivá (ide teda o dôkaz sporom). Keďže však táto cieľová klauzula nemá byť s čím rezolvovaná, tak o platnosti H zatiaľ nie je možné rozhodnúť.

Vzhľadom na prítomnosť štyroch faktov v báze znalostí je možné realizovať niektoré inferenčné kroky. Keďže neexistuje informácia, ktoré z tých pravidiel, ktoré môžu byť použité, je vhodné použiť a ktoré nie, realizujú sa všetky možné hyperrezolvencie.

$$\frac{A \wedge B \wedge C \rightarrow I \qquad \top \rightarrow A,\ \top \rightarrow B,\ \top \rightarrow C}{\top \rightarrow I}$$

$$\frac{A \wedge B \rightarrow E \qquad \top \rightarrow A,\ \top \rightarrow B}{\top \rightarrow E}$$

$$\frac{B \wedge C \rightarrow E \qquad \top \rightarrow B,\ \top \rightarrow C}{\top \rightarrow E}$$

Výsledkom je dokázanie platnosti dvoch nových faktov I a E, ktoré sa pridajú k báze znalostí (aj keď platnosť E bola redundantne odvodená dvakrát, do bázy sa pridá iba jedna kópia príslušného faktu).

Keďže naďalej niet s čím rezolvovať cieľovú klauzulu, je možné vykonať ďalšie kolo inferencií. Aktuálne sa v báze nachádza šesť faktov – štyri pôvodné a dva novo odvodené. Vďaka tomu je možné využiť štyri z pravidiel (vrátane troch pravidiel použitých v predchádzajúcom kole) a opätovne odvodiť platnosť I a E ako aj pomocou

$$\frac{E \wedge F \rightarrow H \qquad \top \rightarrow E,\ \top \rightarrow F}{\top \rightarrow H}$$

odvodiť platnosť nového faktu H. O splnení cieľa sa možno jednoducho presvedčiť, keďže teraz už je možná rezolvencia cieľovej klauzuly. Vykoná sa rezolvencia získaného faktu s negáciou cieľovej hypotézy

$$\frac{\top \rightarrow H \qquad H \rightarrow \bot}{\top \rightarrow \bot}$$

čoho výsledkom je spor (odvodená klauzula neplatí) a teda bola dokázaná platnosť cieľa H.

RIADENIE DOKAZOVANIA

Z uvedeného príkladu je zrejmé, že bolo vykonaných viac inferencií než bolo nutných – napríklad dokázanie platnosti faktu I bolo zbytočné, pretože nijako neprispelo k požadovanému dôkazu H. To preto, pretože odvodzovací mechanizmus nemal k dispozícii žiadnu informáciu, ktorá by ho navigovala smerom k H. Mal jedinú šancu – dokázať všetko čo sa dá. Aj keď sa dopredné reťazenie dá použiť v úlohe *cieľovo orientovaného* dokazovania, tendencia robiť z hľadiska cieľa nepotrebné odvodzovacie kroky ho skôr predurčuje pre dokazovanie *riadené dátami*, pri ktorom sa vychádza z existujúcich faktov a dokazovanie sa realizuje bez prítomnosti nejakého špecifického cieľa.

ALGORITMUS DOPREDNÉHO REŤAZENIA PRAVIDIEL

Spôsob odvodzovania založený na hyperrezolvenčných krokoch je možné formalizovať v tvare algoritmu Alg. 3.1. Daný algoritmus v cykle odvádza nové fakty, pričom toto odvádzanie končí v jednej z dvoch možných situácií – alebo bol odvodený fakt, ktorého dokázanie bolo cieľom odvádzania (riadok 9) alebo už nie je možné odvodiť žiadny nový fakt, ktorý ešte nie je zaradený v znalostnej báze (riadok 11). Algoritmus je teda možné použiť aj v takom prípade, že nie je k dispozícii žiadny cieľový fakt, a spoliehať sa na jeho zastavenie na základe druhej situácie.

V každej iterácii cyklu sa skúmajú všetky pravidlá prítomné v znalostnej báze, ktorých predpokladová LHS časť obsahuje aspoň jeden literál. Tieto pravidlá sa skúmajú na možnosť odvodenia platnosti ich záveru pomocou hyperrezolvenčného kroku. Teda sa overuje, či v znalostnej báze sa

vstup: znalostná báza KB a $Ciel$ ktorý je potrebné dokázať
výstup: doplnená znalostná báza KB

```
    fch(KB, Ciel)
1.      do
2.          novy = {}
3.          foreach (LHS → RHS) ∈ KB do
4.              foreach θ' ∈ {θ | subst(θ, LHS) ≡ subst(θ, l_1 ∧ ... l_n),
5.                                (⊤ → l_i) ∈ KB, i = 1, ..., n} do
6.                  RHS' = subst(θ', RHS)
7.                  novy = novy ∪ (⊤ → RHS')
8.          KB = KB ∪ novy
9.          if (⊤ → Ciel) ∈ novy then return KB
10.     while novy ≠ ∅
11.     return KB
```

Alg. 3.1: Dopredné reťazenie pravidiel

nachádza taký súbor faktov, pomocou ktorého je možné eliminovať všetky negatívne literály klauzuly, reprezentujúcej skúmané pravidlo (teda vlastne či je možné aplikovať odvodzovanie pomocou *modus ponens*).

Ak pre nejaké pravidlo existuje možnosť viacerých hyperrezolvenčných krokov s rôznymi unifikátormi, ktoré vďaka substitúcii vedú na rôzne podoby odvodeného faktu, pridajú sa do znalostnej bázy všetky odvodené podoby záveru pravidla.

PRODUKČNÉ SYSTÉMY

Na doprednom reťazení pravidiel sú založené *produkčné systémy*, ktoré obsahujú dve databázy – *databázu pravidiel* a *databázu faktov*. Tieto databázy je potrebné naplniť konkrétnymi faktami a pravidlami platnými v nejakej vybranej doméne. Takéto systémy potom pracujú v dvoch módoch:

- dávkový režim,
- iteratívny režim.

Dávkový mód reprezentuje jednorazový spôsob použitia – do systému je vložené všetko, o čom je známe že platí v danej doméne. Systém na zá-

```
(deffacts ostrov
    (pravda je poctivec poctivec)
    (pravda je luhar luhar)
    (klamstvo je poctivec luhar)
    (klamstvo je luhar poctivec)
    (ostrovan jano poctivec)
    (ostrovan juro poctivec)
    (ostrovan fero luhar)
)

(defrule povedal-poctivec
    (ostrovan ?meno poctivec)
    (ostrovan ?kto ?kmen)
    (pravda je ?kmen ?co)
=>
    (printout t ?meno " povedal: " ?kto " je " ?co crlf)
    (assert (povedal ?meno je ?kto ?co))
)

(defrule povedal-luhar
    (ostrovan ?meno luhar)
    (ostrovan ?kto ?kmen)
    (klamstvo je ?kmen ?co)
=>
    (printout t ?meno " povedal: " ?kto " je " ?co crlf)
    (assert (povedal ?meno je ?kto ?co))
)
```

Obr. 3.1: Kód v jazyku Clips

klade dopredného reťazenia odvodí všetko, čo sa na základe daného obsahu znalostnej bázy odvodiť dá. V druhom režime sa vždy, keď sa zistí platnosť nejakého faktu na základe vonkajšieho prostredia, tento fakt pridá do databázy faktov a systém spustí inferenciu nových faktov. Keď už systém nevie nič ďalšie dokázať, tak opäť čaká na vloženie nového faktu.

Jedným z reprezentantov produkčných systémov je programovací jazyk *Clips*[2] (C Language Integrated Production System). Jazyk má svoju vlastnú syntax, ktorá je podobná syntaxi predikátového počtu a preto prepis pravidiel vyjadrených v predikátovom počte do tejto syntaxe zachováva čitateľnosť pravidiel. Ukážkou je reprezentácia úlohy 1 z ilustračného prí-

CLIPS

[2] http://clipsrules.sourceforge.net/

kladu Pr. 3.1. Výsledný program v tomto jazyku je na Obr. 3.1 – obsahuje sedem faktov a dve pravidlá, reprezentujúce prístup poctivcov a luhárov.

PRÍPRAVA DATABÁZ JAZYKA CLIPS

Pre vyriešenie úlohy je po zavedení programu (zároveň sa naplní databáza pravidiel obomi pravidlami) potrebné ešte inicializovať systém, keď sa naplní databáza faktov tými faktami, ktoré sú uvedené v konštrukte *deffacts*:

```
CLIPS> (load "ostrov.clp")
Defining deffacts: ostrov
Defining defrule: povedal-poctivec +j+j+j
Defining defrule: povedal-luhar +j+j+j
TRUE
CLIPS> (facts)
CLIPS> (rules)
povedal-poctivec
povedal-luhar
For a total of 2 defrules.
CLIPS> (reset)
CLIPS> (facts)
f-0 (initial-fact)
f-1 (pravda je poctivec poctivec)
f-2 (pravda je luhar luhar)
f-3 (klamstvo je poctivec luhar)
f-4 (klamstvo je luhar poctivec)
f-5 (ostrovan jano poctivec)
f-6 (ostrovan juro poctivec)
f-7 (ostrovan fero luhar)
For a total of 8 facts.
```

kde `CLIPS>` reprezentuje nápoveď, indikujúcu očakávanie príkazu (príkaz `facts` vypisuje obsah databázy faktov, `rules` zase obsah databázy pravidiel a `reset` slúži pre samotnú inicializáciu).

REŤAZENIE V AKCII

Po iniciálnom naplnení oboch databáz je možné získať výpovede obyvateľov ostrova o obyvateľoch pomocou príkazu `run`

```
CLIPS> (run)
fero povedal: fero je poctivec
fero povedal: juro je luhar
fero povedal: jano je luhar
jano povedal: fero je luhar
juro povedal: fero je luhar
juro povedal: juro je poctivec
juro povedal: jano je poctivec
jano povedal: juro je poctivec
jano povedal: jano je poctivec
CLIPS>
```

Rozšírenie: Syntax jazyka Clips

Prehľad syntaxe jazyka Clips

Hlavnými dátovými typmi (okrem ďalších typov ako reťazce a rôzne typy adries) sú čísla a symboly. Číselný typ môže byť celočíselný (napr: *12*, *-15*, *+3*) alebo reálny (napr: *15.09*, *-2.3e-4*). Symbol je sekvencia znakov, pričom je možné použiť číslice, písmená (rozlišujú sa veľké a malé písmená) a niektoré ďalšie znaky (napr: *jeden-dva*, *2_slivky*, *Kto?*).

Fakt je zoznam atomických hodnôt, na ktoré sa je možné odvolávať pozične (usporiadané fakty) alebo pomocou mena (šablónové fakty). Prvý typ je jednoduchší – obsahuje jedno alebo viac polí (prvé musí byť symbolom), ktoré sú uzavreté v zátvorkách. Príkladom sú fakty: *(suroviny maslo voda chlieb)*, *(pondelok)* a *(teplota 42)*. Fakty môžu byť dynamicky pridávané (príkaz *assert*), odstránené (príkaz *retract*) alebo modifikované (príkaz *modify*). Pri pridávaní fakt sa nepridá, ak už rovnaký fakt existuje v databáze faktov. Odstránenie a modifikácia vyžaduje identifikáciu faktu napríklad pomocou jeho adresy. Pre inicializáciu databázy faktov slúži konštrukt *deffacts* – pri použití príkazu *reset* sa kolekcia faktov, ktorú konštrukt obsahuje, vloží do databázy.

Pravidlá sa definujú pomocou konštruktu *defrule*, ktorého tvar je

```
(defrule <meno pravidla>
    <LHS>
=>
    <RHS>
)
```

kde *<meno pravidla>* musí byť symbolom, *<LHS>* reprezentuje podmienky pre odpálenie pravidla a *<RHS>* zase akcie, ktoré sa majú vykonať pri odpálení pravidla. Akcie môžu byť v zásade dvojakého typu – alebo menia obsah databázy faktov alebo vykonávajú inú činnosť.

Základným typom podmienky je vzor. Vzor má tvar faktu s tým, že môže obsahovať premenné. Premenná má tvar symbolu, začínajúceho znakom *?* alebo dvojicou znakov *$?* – prvý typ premennej reprezentuje obsah jedného poľa, zatiaľ čo druhý typ reprezentuje obsah viacerých polí. Príkladom vzoru je *(teplota ?x)* či *(suroviny $?co)*. K premenným je možné pridávať ohraničenia rôznych typov, príkladom môžu byť vzory *(teplota ?x&:(>?x 37)&~?vcera)* či *(teplota ?x&39|40|41)*.

Iným typom podmienky je testovacia podmienka, ktorej príkladom je *(test (>?teplota 40))*. Je možné vytvárať aj zložitejšie podmienky pomocou negovania podmienok či ich združovania pomocou konjunkcie alebo disjunkcie, napríklad *(and (cislo ?x) (not (cislo ?y&:(>?y ?x))))*.

Zobrazené výpisy sú výsledkami série hyperrezolvencií oboch pravidiel z bázy pravidiel.

USPORIADANIE HYPERREZOLVENCIÍ

Samotný algoritmus Alg. 3.1 hovorí o spracovaní všetkých pravidiel a vyhľadaní všetkých hyperrezolvencií, avšak nehovorí nič o ich usporiadaní – v akom poradí sa jednotlivé pravidlá či hyperrezolvencie majú uvažovať. Pri logickom systéme každý fakt alebo platí alebo neplatí – a je jedno, či platnosť daného faktu je odvádzaná ako prvá alebo ako posledná . Produkčný systém však implicitne uvažuje aj s plynúcim časom – nejaký fakt môže (v nejakom čase) platiť a zároveň (v nejakom inom čase) neplatiť. Z tohto vyplývajú dva dôsledky. Jedným je to, že Clips má nástroje nielen na pridanie nového faktu do znalostnej bázy ale aj na odstránenie platného faktu z tejto znalostnej bázy. Pre ilustráciu druhého dôsledku uvažujme sadu pravidiel

$$\begin{aligned} P_1 &: \mathit{volne\_sedadlo}(S) \wedge \mathit{stoji}(X) \rightarrow \mathit{sedi}(X,S) \\ P_2 &: \mathit{sedi}(X,S) \rightarrow \neg\, \mathit{volne\_sedadlo}(S) \\ P_3 &: \mathit{sedi}(X,S) \rightarrow \neg\, \mathit{stoji}(X) \end{aligned}$$

a nasledovné fakty

$$\begin{array}{ccccc} \mathit{volne\_sedadlo}(s7) & & & & \mathit{volne\_sedadlo}(s9) \\ & \mathit{stoji}(peter) & \mathit{stoji}(karol) & \mathit{stoji}(jano) & \end{array}$$

Je zrejmé, že v danej situácii si všetci cestujúci môžu sadnúť na voľné sedadlo, keďže pre prvé pravidlo existuje šesť možností hyperrezolvencie s unifikátormi $\theta_1 = \{X/peter, S/s7\}$ až $\theta_6 = \{X/jano, S/s9\}$. Problém je však v tom, že nie je možné realizovať všetky možnosti súčasne, pretože kapacita každého sedadla je obmedzená – akonáhle si jeden z nich sadne na nejaké, toto sedadlo prestáva byť voľné. A zároveň nikto si nemôže sadnúť na viac ako jedno sedadlo, pretože po sadnutí na jedno prestáva stáť.

Aby toto intuitívne chápanie bolo v súlade s výsledkom dopredného reťazenia v produkčnom systéme, je potrebná zmena v činnosti produkčného systému oproti algoritmu Alg. 3.1. Akonáhle sa realizuje prvá hyperrezolvencia pravidla P_1 (ktorá jedného z cestujúcich posadí na sedadlo), je nutné prejsť na hyperrezolvencie pravidiel P_2 a P_3 (zabezpečujúcich informáciu o zmene stavu obsadenosti daného sedadla a zmene stavu daného cestujúceho).

PRIORITA PRAVIDIEL

Teda spracovanie niektorých pravidiel je potrebné uprednostniť pred spracovaním pravidiel iných. Toto je v produkčných systémoch riešené zavedením priority pravidiel – každé pravidlo má v programe priradenú určitú prioritu (pri neuvedení priority sa mu automaticky priradí určitá preddefinovaná priorita – všetky pravidlá bez explicitného priradenia priority budú

mať rovnakú prioritu). Pri realizácii pravidiel sa jednotlivé pravidlá spracovávajú (cyklus na riadkoch 3 až 7 v algoritme Alg. 3.1) podľa ich priority – na pravidlá určitej priority sa prechádza až potom, keď boli preskúmané všetky pravidlá s vyššou prioritou. V rámci pravidiel s rovnakou prioritou sa používa usporiadanie podľa nejakej v systéme zabudovanej stratégie[3].

USPORIADANIE UNIFIKÁTOROV

Pravidlo P_1 dokáže usadiť jedného cestujúceho na sedadlo $s7$. O tom, kto to bude, rozhoduje to, či sa bude skôr realizovať hyperrezolvencia s unifikátorom $\theta_1 = \{X/peter, S/s7\}$ alebo s unifikátorom $\theta_2 = \{X/karol, S/s7\}$ či s $\theta_3 = \{X/jano, S/s7\}$. Teda navyše k usporiadaniu pravidiel pribúda aj usporiadanie jednotlivých unifikátorov, ktoré určuje možné poradie hyperrezolvencií (vnútorný cyklus algoritmu Alg. 3.1 na riadkoch 4 až 7). Toto usporiadanie produkčný systém typicky realizuje na základe nejakej zabudovanej stratégie (riadenej napr. usporiadaním faktov v databáze faktov).

Aj keď v našej ukážke na začiatku pre prioritnejšie pravidlá P_2 a P_3 nie je možná žiadna hyperrezolvencia zatiaľ čo pre pravidlo P_1 ich je možných niekoľko, vykoná sa iba jedna z nich – pretože jej vykonanie umožní hyperrezolvencie pravidiel P_2 a P_3, ktoré na oplátku zrušia niektoré z už pripravených hyperrezolvencií pravidla P_1. Aby bolo možné toto dosiahnuť, tak v modifikovanej podobe Alg. 3.1 v jednej iterácii sa vždy realizuje iba jedna hyperrezolvencia (s unifikátorom, ktorý je prvý v poradí) jedného pravidla (ktoré je najprioritnejšie z tých, pre ktoré je možné hyperrezolvenciu vykonať).

ČINNOSŤ PS

Na základe uvedeného je možné vytvoriť algoritmus práce produkčného systému. Tento algoritmus je uvedený v Alg. 3.2, pričom sú v ňom použité termíny, používané v oblasti produkčných systémov. Jedná sa o tri doposiaľ neznáme termíny: *aktivácia*, *agenda* a *odpálenie pravidla*.

AKTIVÁCIA PRAVIDLA

Pod *aktiváciou pravidla* sa rozumie vytvorenie dvojice tvaru

$$(LHS \rightarrow RHS, \theta)$$

indikujúcej, že na základe faktov, ktoré sa aktuálne nachádzajú v znalostnej báze KB je pre nejaké pravidlo tvaru $LHS \rightarrow RHS$ možné vykonať hyperrezolvenciu, pričom pre túto je nutné použiť unifikátor θ (vytvorený na základe unifikácie LHS pravidla a faktov v KB). Ak je teda pre nejaké pravidlo možné vykonať viac rôznych hyperrezolvencií, ktoré sa líšia použitým unifikátorom, potom tomuto pravidlu prislúchajú viaceré aktivácie – pre každý unifikátor jedna.

[3] Clips umožňuje používať sedem stratégií usporiadania, napríklad usporiadanie do šírky, do hĺbky, náhodné, podľa zložitosti, atď. Implicitne sa používa stratégia usporiadania do hĺbky, keď nové aktivácie sa umiestnia do agendy pred aktivácie s rovnakou prioritou.

vstup: znalostná báza KB
výstup: doplnená znalostná báza KB

1. aktivácia pravidiel z KB a ich uloženie do *Agendy*
2. usporiadanie aktivácií v *Agende*
3. odpálenie prvej aktivácie
4. odstránenie neplatných aktivácií z *Agendy*
5. pridanie nových aktivácií do *Agendy*
6. **if** *Agenda* $\neq \emptyset$ **then** *goto* 2

Alg. 3.2: Schéma činnosti produkčného systému

Pre uchovávanie aktivácií sa používa *agenda* – dátová štruktúra, umožňujúca potrebné manipulácie s jednotlivými aktiváciami. Na začiatku je agenda prázdna a preto je ju nutné iniciálne naplniť možnými aktiváciami (krok 1 algoritmu Alg. 3.2). Následne sú všetky aktivácie v agende usporiadané (krok 2) podľa poradia, ktoré rešpektuje prioritu pravidiel a stratégiu usporiadania pravidiel s rovnakou prioritou – aktivácie prioritnejších pravidiel sú uvedené pred aktiváciami pravidiel s nižšou prioritou. Aj keď takýmto spôsobom je možné dosiahnuť úplné usporiadanie aktivácií, dôležité je iba to, ktorá z aktivácií sa nachádza na začiatku usporiadania. Táto aktivácia sa vyberie z agendy a realizuje sa (krok 3).

ODPÁLENIE PRAVIDLA

Hovoríme že dochádza k *odpáleniu* pravidla – vykoná sa hyperrezolvencia pravidla z vybranej aktivácie použitím unifikátora z tejto aktivácie, teda dochádza k realizácii $subst(\theta, RHS)$. Realizácia pravej časti odpáleného pravidla môže mať za následok zmenu KB, a to ako pridanie nových faktov tak aj odstránenie už existujúcich faktov. Keďže zmena KB môže mať za následok, že niektorá hyperrezolvencia, ktorá pred odpálením pravidla bola možná (a teda zodpovedajúca aktivácia je uložená v agende) už možná nie je, tak je nutné príslušnú aktiváciu z agendy odstrániť (krok 4). Na druhej strane zmena KB môže mať za následok, že nejaká hyperrezolvencia, ktorá pred odpálením pravidla možná nebola, už možná je (ale zodpovedajúca aktivácia sa v agende nenachádza), takže je nutné príslušnú aktiváciu do agendy doplniť (krok 5). Po týchto krokoch je agenda aktu-

alizovaná a zostáva už iba opäť aktivácie v nej usporiadať aby bolo možné odpáliť ďalšie pravidlo. V prípade, že sa agenda vyprázdni a teda už neobsahuje žiadnu aktiváciu, tak sa činnosť produkčného systému zastaví.

Porovnávanie vzorov

Hlavnou operáciou, ktorá je vykonávaná produkčným systémom, je *porovnávanie vzorov* voči databáze vzorov – na tejto operácii je založené vytváranie nových aktivácií ako aj rušenie neplatných aktivácií. Toto porovnávanie je výpočtovo veľmi náročné, pretože je potrebné porovnať každý vzor (podmienku) každého pravidla voči každému faktu. Priama implementácia tohto porovnávania tak vedie na tri vnorené cykly, ktoré je potrebné realizovať po každej zmene databázy faktov.

Reteho sieť

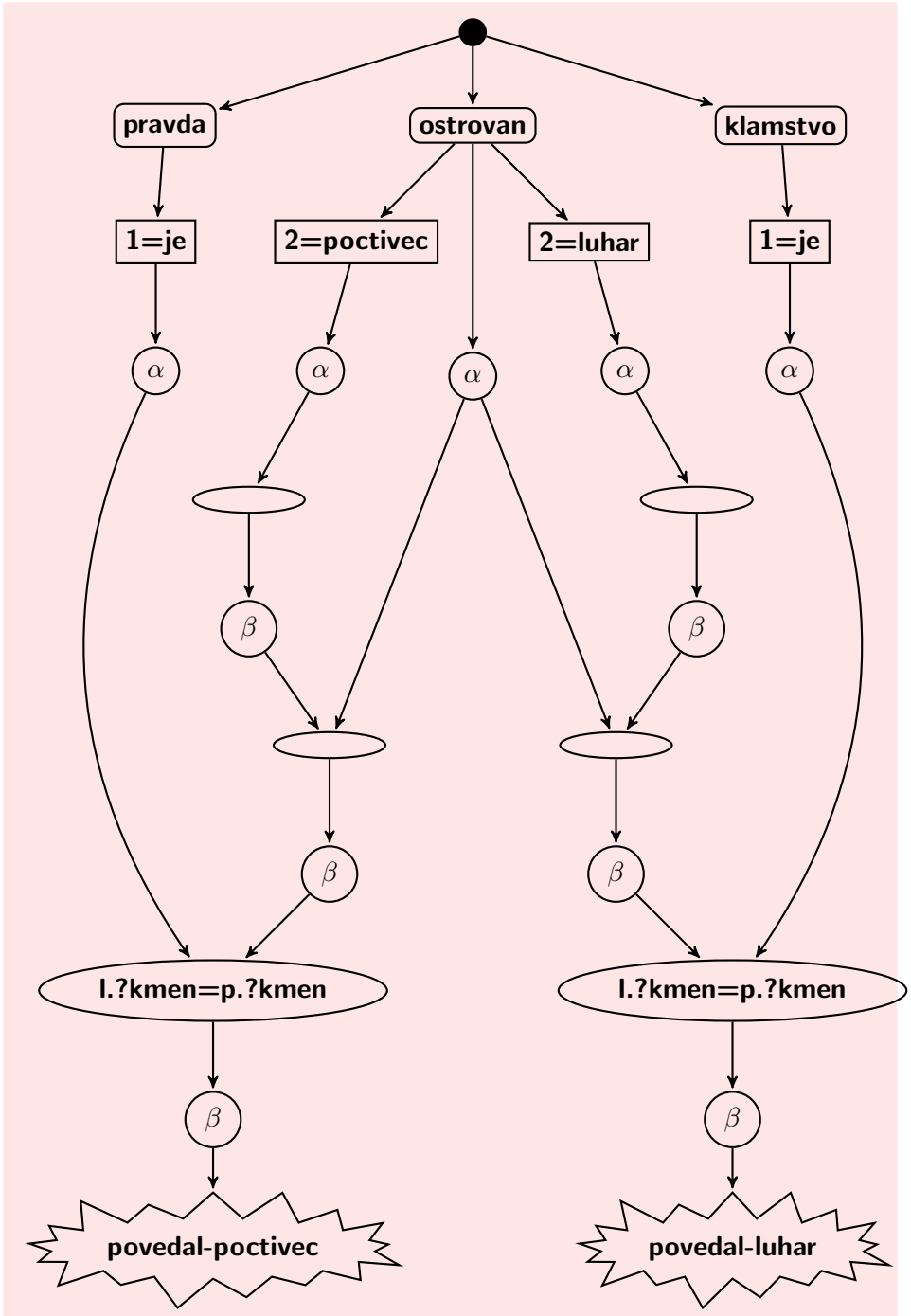
Pre minimalizáciu potrebných porovnávaní je možné pravidlá skompilovať do dátovej štruktúry označovanej ako *Reteho sieť*. Takáto sieť zodpovedajúca dvom pravidlám z Obr. 3.1 je zobrazená na Obr. 3.2. Jedná sa vlastne o orientovaný acyklický graf s viacerými typmi uzlov, ktoré majú jeden alebo dva vstupy a jeden výstup.

Šírenie faktov v Reteho sieti

Aktualizácia aktivácií je realizovaná ako šírenie faktov touto vytvorenou sieťou. Všetky fakty, ktoré sa aktuálne nachádzajú v databáze faktov, sú pamätané v sieti. Každý fakt sa pri svojom vkladaní šíri sieťou pozdĺž hrán v smere ich orientácie, pričom v sieti je pamätaný vo vybraných uzloch. Pri svojom postupe sú fakty kombinované s inými faktami (tieto kombinácie sú tiež pamätané vo vybraných uzloch). Fakty alebo ich kombinácie postupujú sieťou do rôznej hĺbky – ak sa nejakej kombinácii podarí prejsť celou sieťou, potom táto kombinácia reprezentuje novú aktiváciu.

Preto pri aktualizácii aktivácií nie je nutné iterovať cez fakty v databáze faktov, pretože stačí využívať stopy jednotlivých faktov v sieti – opakované iterovanie bolo nahradené uchovávaním informácií. Iba v prípade zmien v databáze faktov je potrebné spracovávať tieto zmeny – ak bol pridaný fakt, potom tento fakt je šírený sieťou a informácia o ňom je pridávaná do siete, alebo ak bol fakt vypustený, potom je šírený sieťou a informácia o ňom je zo siete vymazávaná. Taktiež už nie je potrebné iterovať cez všetky podmienky všetkých pravidiel. Je to preto, pretože fakt sa nešíri všetkými hranami siete – sieť má charakter triediacej štruktúry, ktorá fakt prepúšťa iba pozdĺž niektorých hrán. Toto prepúšťanie sa deje iba do obmedzenej hĺbky, pretože akonáhle je detekovaná zbytočnosť šírenia faktu pozdĺž nejakej hrany, tak je šírenie daného faktu pozdĺž tejto hrany zastavené. Iterácia cez všetky podmienky všetkých pravidiel je teda nahrádzaná kontrolou iba niektorých podmienok niektorých pravidiel. Teda zvýšenie rýchlosti (odstránenie opakovaných iterácií) pri aktualizáciách aktivácií bolo zaplatené spotrebou pamäti pre uchovávanie siete a v nej pamätaných informácií.

Reteho sieť obsahuje uzly rôzneho typu: *koreňové*, *typové*, *selekčné*, *spoj-*



Obr. 3.2: Reteho sieť

kové, *pravidlové* a dva typy *pamäťových* uzlov. Úloha pamäťových uzlov je zrejmá – slúžia pre pamätanie informácií o faktoch alebo ich kombináciách. Koreňové a pravidlové uzly sú identifikačnými uzlami, identifikujú "začiatok" a "koniec" siete. Ostatné uzly sú podmienkovými (filtračnými) uzlami – rozhodujú o tom, či fakty sa budú šíriť cez daný uzol ďalej pozdĺž hrán alebo budú zablokované.

UZLY V RETEHO SIETI

Koreňový uzol označuje vstupný bod do siete, teda nie je v sieti taká hrana, ktorá by doň vstupovala. Uzol takéhoto typu je v sieti vždy iba jeden a nemá žiadny účinok na fakty – každý fakt môže cez tento uzol prejsť a postupovať pozdĺž hrán ďalej. Ak predpokladáme, že na vstup siete prišli nasledujúce fakty

KOREŇOVÉ UZLY

```
f-1: (pravda je poctivec poctivec)
f-2: (pravda je luhar luhar)
f-6: (ostrovan juro poctivec)
f-7: (ostrovan fero luhar)
```

tak všetky budú pokračovať každou z troch výstupných hrán koreňového uzla.

Po koreňovom uzle nasledujú *typové* uzly (na obrázku 3.2 obdĺžniky so zaoblenými hranami). Každý z nich reprezentuje iný typ faktu, pričom pod typom faktu sa rozumie symbol umiestnený na začiatočnom poli faktu (označovanom ako nultá pozícia). Jedná sa o filtrovací uzol – zo svojho vstupu prepustí na svoj výstup iba tie fakty, ktorých typ je zhodný s typom, reprezentovaným daným uzlom. Teda typový uzol "pravda" prepustí iba fakty f-1 a f-2, typový uzol "ostrovan" prepustí iba fakty f-6 a f-7 a typový uzol "klamstvo" zablokuje všetky štyri fakty a neprepustí ďalej žiadny z nich.

TYPOVÉ UZLY

Nasledujú *selekčné* uzly (na 3.2 obdĺžniky s ostrými hranami). Každý selekčný uzol je zameraný na testovanie jednej vnútrovzorovej podmienky – podmienky stanovujúcej nejaké obmedzenie v rámci jedného vzoru. Ak by sme mali napríklad vzor *(teplota rano ?x vecer ?x)*, tak v tomto vzore existujú tri obmedzenia:

SELEKČNÉ UZLY

- hodnota na prvej pozícii je obmedzená na symbol *rano*,
- hodnota na tretej pozícii musí byť symbolom *vecer*,
- hodnota na druhej pozícii musí byť rovnaká ako na štvrtej pozícii.

Ak vzor obsahuje viacero obmedzení, tak potom je viacero selekčných uzlov zoradených za sebou. Ak vo vzore nie je žiadne obmedzenie, potom tomuto vzoru neprislúcha žiadny selekčný uzol. Sieť je kompilovaná takým spôsobom, aby bolo uzlov čo najmenej – teda jeden selekčný uzol môže reprezentovať nejakú podmienku, ktorá sa nachádza súčasne vo viacerých vzoroch

či už toho istého alebo rôznych pravidiel. Takýmto spôsobom dochádza k zdieľaniu obmedzení. Selekčný uzol filtruje tie fakty, ktoré nespĺňajú obmedzenie reprezentované uzlom. Preto uzol "2=poctivec" (reprezentujúci požiadavku, aby na druhej pozícii bol uvedený symbol) prepustí fakt f-6 ale zablokuje fakt f-7.

α PAMÄŤOVÉ UZLY

Ak nejaký fakt z pohľadu nejakého vzoru splnil všetky vnútrovzorové obmedzenia, je potrebné si ho pre daný vzor zapamätať, pretože aj ak aktuálne nie je kompatibilný s inými vzormi, v budúcnosti sa takým môže stať. K tomuto slúžia tzv. *alfa pamäťové* uzly. Každý takýto uzol si pamätá všetky fakty, ktorým sa podarilo prejsť cez sieť až k danému uzlu, pričom zároveň tým faktom dovoľuje šíriť sa ďalej. V našom prípade najľavejší alfa pamäťový uzol si bude pamätať fakty f-1 a f-2, jeho sused sprava si pamätá f-6, stredný alfa uzol si pamätá dva fakty f-6 a f-7, druhý uzol sprava si pamätá f-7 a najpravejší pamäťový uzol ostáva prázdny. Vrstva alfa uzlov delí sieť na dve podsiete:

- *alfa sieť* venovanú jednotlivým vzorom uvažovaným osobitne,
- *beta sieť* zameranú na kombinovanie vzorov do LHS jednotlivých pravidiel.

SPOJKOVÉ UZLY

Pre kombinovanie vzorov je dôležité testovanie medzivzorových obmedzení stanovujúcich, kedy je možné fakty považovať za kompatibilné. Ak by sme mali napríklad dva vzory *(teplota ?meno ?teplota&:(< ?teplota 37))* a *(stav ?meno zdravy)*, tak pre túto dvojicu existuje jedna medzivzorová podmienka – hodnota na prvej pozícii faktu, spĺňajúceho prvý vzor, musí byť rovnaká ako hodnota na prvej pozícii faktu, spĺňajúceho druhý vzor. Ak toto platí, potom oba fakty sú z pohľadu danej dvojice vzorov kompatibilné. Pre kombinovanie faktov slúžia *spojkové* uzly (na obrázku 3.2 elipsy). Spojkový uzol reprezentuje obmedzenia medzi dvojicou vzorov – tých obmedzení môže byť viac alebo na druhej strane ani jedno. Spojkový uzol sa však dá chápať nie ako spájanie dvoch vzorov, ale ako pripájanie vzoru k predchádzajúcim vzorom. Keďže však prvý vzor v LHS pravidla niet k čomu predchádzajúcemu pripojiť, tak v sieti budú vystupovať aj spojkové uzly iba s jedným vstupom.

β PAMÄŤOVÉ UZLY

Aj keď nejaký fakt (alebo kombinácia faktov) aktuálne nevie prejsť cez spojkový uzol kvôli neprítomnosti vhodného doplnku na druhom vstupe spojkového uzla, to sa môže v budúcnosti zmeniť. Preto musí ostať na vstupe spojkového uzla a čakať na svoju príležitosť. Pre tento účel slúžia nielen alfa pamäťové uzly, ale aj *beta pamäťové* uzly pamätajúce si kombinácie faktov. Takýto pamäťový uzol sa nachádza na výstupe každého spojkového uzla.

V našom prípade ľavý spojkový uzol s dvomi vstupmi avšak bez podmienky má na ľavom vstupe fakt f-6 (obsah predradenej beta pamäti) a na pravom vstupe fakty f-6 a f-7 (obsah alfa pamätí na ktorú daný spojkový uzol nadväzuje). Keďže uzol nereprezentuje žiadne obmedzenie, tak na jeho výstupe budú dve kombinácie (f-6,f-6) a (f-6,f-7). Analogicky ľavý spojkový uzol s podmienkou (požadujúcou aby hodnota premennej *?kmen* v ľavom vstupe bola rovnaká ako hodnota premennej *?kmen* v pravom vstupe) má na svojom ľavom vstupe fakty f-1 a f-2 (obsah predchádzajúcej alfa pamäti) a na svojom pravom vstupe (obsah predradenej beta pamäti) kombinácie faktov (f-6,f-6) a (f-6,f-7). Na jeho výstupe teda budú kombinácie (f-1,f-6,f-6) a (f-2,f-6,f-7).

PRAVIDLOVÉ UZLY

Pomocou spojkových uzlov sa spájajú iba vzory, ktoré sa nachádzajú spolu v LHS toho istého pravidla. Na spojenie n vzorov je potom potrebných n spojkových uzlov – každý z nich pripája niektorý zo vzorov k spojeniu predchádzajúcich vzorov[4]. Na výstup posledného spojkového uzla nadväzuje beta pamäťový uzol, ktorý bude uchovávať všetky kombinácie faktov, pomocou ktorých sa dá splniť predpokladová časť nejakého pravidla. A na tento uzol záverečne nadväzuje *pravidlový* uzol, ktorý toto pravidlo reprezentuje. V našom príklade je teda z aktuálneho stavu siete zrejmé, že pravidlo "povedal-poctivec" má dve aktivácie, pričom jedna je založená na faktoch f-1, f-6 a f-6 (juro vypovedá o sebe) a druhá na f-2, f-6 a f-7 (juro vypovedá o ferovi).

Reteho sieť sa buduje v okamihu zavádzania pravidiel do systému Clips. Systém o tom podáva informáciu nasledovným výpisom zodpovedajúcim programu z Obr. 3.1

```
CLIPS> (load "ostrov.clp")
Defining deffacts: ostrov
Defining defrule: povedal-poctivec +j+j+j
Defining defrule: povedal-luhar +j+j+j
TRUE
```

kde pri budovaní vnútornej štruktúry pri kompilovaní jednotlivých pravidiel "+j" oznamuje vytvorenie nového spojkového (join) uzla zatiaľ čo znovu použitie spojkového uzla, ktorý už je vytvorený, je indikované prostredníctvom "=j". Z výpisu je teda vidieť, že pre každé z pravidiel bolo potrebné vytvoriť tri spojkové uzly, čo zodpovedá štruktúre siete na Obr. 3.2.

[4]Clips sa v tomto líši od štandardnej topológie spojkových uzlov, kde na spojenie n vzorov je potrebných iba $n-1$ spojkových uzlov, pretože prvý spojkový uzol má tiež dva vstupy, keďže spája prvé dva vzory.

3.2 Spätné reťazenie pravidiel

SPÄTNÉ POUŽITIE PRAVIDIEL

Algoritmus spätného reťazenia využíva pravidlá spätným spôsobom – postupuje od záverov k predpokladom. Podľa tohto prístupu sa o platnosti nejakého cieľového predikátu dá rozhodnúť dvomi spôsobmi:

- je známe, že daný predikát platí (vo forme existujúceho faktu), takže nie je potrebné nič dokazovať,
- použije sa nejaké pravidlo, ktoré dokazuje platnosť daného predikátu (pravidlo tú platnosť dokazuje svojim záverom).

Pri použití pravidla sa tak dokazovanie platnosti hlavy pravidla nahrádza dokazovaním platnosti tela pravidla. Každý literál, ktorý vytvára toto telo, sa stáva novým cieľom, ktorého platnosť je potrebné dokázať – napríklad použitím iného pravidla. Takýmto spôsobom je možné pravidlá reťaziť. Reťazenie končí v situácii, keď platnosť literálu je potvrdená pomocou faktu alebo neexistuje žiadne pravidlo, ktorého hlava by mohla potvrdiť platnosť tohto literálu.

REZOLVENČNÉ PRAVIDLO

Vzhľadom na to, že sa používajú iba také klauzuly, ktoré reprezentujú implikačné pravidlá, je možné uvažovať iba špeciálny prípad rezolvenčného pravidla, kde do rezolvencie vstupujú dve implikácie a výstupom je jedna implikácia. Toto bude mať tvar

$$\frac{A \wedge B \wedge C \rightarrow Q' \qquad P \wedge Q \wedge R \rightarrow S}{subst(\theta, P \wedge A \wedge B \wedge C \wedge R \rightarrow S)}$$

kde literál Q v tele druhého pravidla, ktorý je komplementárny s hlavou prvého pravidla (teda platí $\theta = unify(Q', Q)$), je v tele druhého pravidla nahradený telom pravidla prvého. Rezolvencia dvoch Hornových klauzúl ústi vždy do výslednej Hornovej klauzuly. Keďže pri predikátovom počte prvého rádu literály môžu obsahovať aj premenné, tak hlavu jedného pravidla je potrebné unifikovať s príslušným literálom v tele druhého pravidla a výsledný unifikátor θ aplikovať na všetky ostávajúce literály.

Pre ilustráciu uvažujme existenciu znalostnej bázy, ktorá obsahuje týchto šesť pravidiel

$$\begin{array}{lll} A \wedge B \rightarrow E & C \wedge E \rightarrow F & E \wedge D \rightarrow F \\ C \wedge D \rightarrow G & I \wedge F \rightarrow J & I \wedge G \rightarrow J \end{array}$$

a štyri fakty A, B, D a I. Cieľom je dokázať platnosť J.

Keďže pri rezolvenčnom spôsobe dokazovania sporom sa vychádza z negácie cieľa, tak teraz začíname s pravidlom $J \rightarrow \bot$, ktoré túto negáciu reprezentuje.

Pri rezolvencii tohto pravidla s pravidlom $I \wedge F \to J$ bude J nahradené a výsledkom bude pravidlo $I \wedge F \to \bot$. Postupným rezolvovaním s inými pravidlami z bázy KB sa získa

$$\dfrac{\dfrac{\dfrac{J \to \bot \qquad I \wedge F \to J}{I \wedge F \to \bot} \qquad E \wedge D \to F}{I \wedge E \wedge D \to \bot} \qquad A \wedge B \to E}{I \wedge A \wedge B \wedge D \to \bot}$$

kde telo výsledného pravidla obsahuje iba také literály, ktoré nie je možné dokázať žiadnym pravidlom, pretože žiadne pravidlo v KB nemá žiadny z týchto literálov vo svojej hlave.

Okrem pravidiel sa však v KB vyskytujú ešte aj fakty. Keďže platnosť faktu I sa dá vyjadriť vo forme pravidla $\top \to I$, tak je možná rezolvencia

$$\dfrac{I \wedge A \wedge B \wedge D \to \bot \qquad \top \to I}{\top \wedge A \wedge B \wedge D \to \bot}$$

pričom následne možno využiť ekvivalenciu $\top \wedge X \equiv X$. Zohľadnením ďalších faktov je možné pokračovať v postupných rezolvenciách

$$\dfrac{\dfrac{\dfrac{A \wedge B \wedge D \to \bot \quad \top \to A}{B \wedge D \to \bot} \qquad \top \to B}{D \to \bot} \qquad \top \to D}{\top \to \bot}$$

Vzhľadom na to, že pravidlo $\top \to \bot$ nie je pravdivé (pretože $\neg\top \vee \bot \equiv \bot$), tak toto posledné pravidlo reprezentuje spor a teda bola dokázaná platnosť cieľa J.

VSTUPNÁ REZOLVENCIA

Uvedený príklad v oblasti výberu klauzúl do úlohy vstupných klauzúl pre rezolvencie ukazuje jednu zaujímavú skutočnosť – do každej zo série rezolvencií vstupovala niektorá z klauzúl, vyskytujúcich sa v KB (či už reprezentujúca pravidlo alebo fakt). Druhá zo vstupných klauzúl alebo reprezentovala cieľ dokazovania (prvá rezolvencia v sérii uvedených rezolvencií) alebo bola medziproduktom, vzniknutým ako výsledok predchádzajúcej rezolvencie. Takáto stratégia výberu klauzúl sa označuje ako stratégia *vstupnej rezolvencie* – obmedzuje jednu zo vstupných klauzúl iba na tie, ktoré sa vyskytujú v báze KB.

Vo všeobecnosti sa nejedná o úplnú stratégiu, môže nastať situácia, keď pomocou nej nie je možné dosiahnuť spor napriek tomu, že množina klauzúl je protirečivá. Ilustráciou takéhoto prípadu je znalostná báza, obsahujúca štyri klauzuly

$$\begin{array}{ll} \neg A \vee \neg B & \neg A \vee B \\ A \vee C & A \vee \neg C \end{array}$$

vstup: znalostná báza KB a $Ciel$ ktorý je potrebné dokázať
výstup: množina dôkazov vo forme unifikátorov

```
    bch(KB, Ciel)
1.      return bch_aux(KB, [Ciel], {})

    bch_aux(KB, Ciele, θ)
1.      odpovede = {}
2.      if Ciele = [ ] then return {θ}
3.      [Prvy|Ostatne] = Ciele
4.      Q = subst(θ, Prvy)
5.      for (LHS → RHS) ∈ vyber(KB, Q) do
6.          θ' = unify(Q, RHS)
7.          if LHS = ⊤
8.          then NoveCiele = Ostatne
9.          else NoveCiele = [LHS|Ostatne]
10.         odpovede = odpovede ∪
                              bch_aux(KB, NoveCiele, comp(θ', θ))
11.     return odpovede

    vyber(KB, Ciel)
1.      P = {}
2.      for (LHS → RHS) ∈ KB do
3.          if unify(Ciel, RHS) ≠ □ then P = P ∪ (LHS → RHS)
4.      return P
```

Alg. 3.3: Spätné reťazenie pravidiel

ktorých protirečivosť nie je možné pomocou tejto stratégie dokázať. Avšak v prípade, že všetky klauzuly majú tvar Hornových klauzúl, sa jedná o úplnú stratégiu. V tomto prípade je možné využiť efektívnosť tejto stratégie, pretože nie je nutné kombinovať výsledky rôznych rezolvencií navzájom, čím sa dosiahne značné orezanie priestoru rezolvencií, ktoré pripadajú do úvahy pre preskúmanie.

ALGORITMUS SPÄTNÉHO REŤAZENIA PRAVIDIEL

Hľadanie protirečivosti klauzúl, spôsobenej pridaním negácie dokazovaného cieľa k pravidlám znalostnej bázy, možno realizovať na základe využitia princípu *prehľadávania do hĺbky* (ak množina Hornových klauzúl má v sebe protirečenie, potom použitie prehľadávania do hĺbky garantuje nájdenie prázdnej klauzuly). Takýmto algoritmom je aj Alg. 3.3. Tento algoritmus sa vlastne systematickým spôsobom snaží vyberať a aplikovať pravidlá, nachádzajúce sa v znalostnej báze, pre dosiahnutie sporu.

V každom kroku vlastne pracuje s pravidlom tvaru $Ciele \rightarrow \perp$, pričom explicitne reprezentuje iba $Ciele$ (čo je vlastne zoznam cieľov, ktoré je potrebné dokázať), ktoré sú vstupom funkcie bch_aux. Literály, ktoré vytvárajú ciele, sa snaží rezolvovať v tom poradí, v akom sú uvedené v zozname, pričom vždy začína aktuálne prvým literálom. Pri úspešnej rezolvencii tohto literálu pomocou faktu je literál vynechaný (riadok 8 algoritmu 3.3) a prechádza sa na literál, ktorý bol druhým cieľom v poradí (rekurzívne volanie na riadku 10). Pri úspešnej rezolvencii pomocou nejakého pravidla sa literály nachádzajúce sa v tele tohto pravidla priradia k aktuálnym cieľom (riadok 9) a rekurzívnym volaním sa prechádza na prvý literál z novo pridaných cieľov (riadok 10).

Pre rezolvenciu každého literálu, ktorý je súčasťou cieľov, sú skúšané všetky fakty a všetky pravidlá zo znalostnej bázy, ktoré sú s ním unifikovateľné (cyklus na riadkoch 5 až 10), pričom samotný výber týchto faktov a pravidiel je realizovaný funkciou *vyber*. Keďže hľadanie neskončí pri prvej úspešnej rezolvencii ale sú skúšané všetky možnosti, znamená to, že uvedený algoritmus spätného reťazenia hľadá všetky možné spôsoby, ako dospieť k sporu a neuspokojí sa iba s jedným z možných spôsobov.

KOMPOZÍCIA UNIFIKÁTOROV

Každý spôsob dosiahnutia sporu je charakterizovaný unifikáciami, ktoré bolo potrebné vykonať počas série rezolvencií vedúcich k tomuto sporu. Unifikátory sú akumulované v premennej *odpovede* a vrátené ako výsledok činnosti algoritmu. Keďže dosiahnutie sporu vyžaduje viacero unifikácií (pre každý z predpokladov jednu), vytvorené unifikátory sú komponované do jedného unifikátora. Pridanie nejakého nového unifikátora θ' k už existujúcemu unifikátoru θ sa deje pomocou kompozičnej funkcie *comp*, pričom účinok kompozície dvoch unifikátorov je z hľadiska substitúcie rovnaký ako keby sa dve substitúcie vykonávali jedna po druhej:

$$subst(comp(\theta_1, \theta_2), X) \equiv subst(\theta_1, subst(\theta_2, X))$$

PROLOG

Spätné reťazenie pravidiel je základom, na ktorom je založený programovací jazyk Prolog[5] – najznámejší reprezentant oblasti *logického programovania*. Prolog obsahuje databázu, ktorú je potrebné naplniť množinou

[5] Jedna z populárnych implementácií jazyka je na http://www.swi-prolog.org/

Rozšírenie: Syntax jazyka Prolog

PREHĽAD SYNTAXE JAZYKA PROLOG

Prolog pozná tri základné typy termov (okrem ďalších typov ako napr. reťazce, bloby): konštanty, premenné a štruktúry. Konštanty slúžia pre pomenovanie individuálnych objektov. Keďže pomenovanie môže byť kvantitatívne alebo kvalitatívne, tak konštanty sa delia na atómy a čísla. Číselná konštanta vyjadruje číslo nejakého typu (napr: *8*, *-3.14*, *1e-10*). Atóm je textovou konštantou, ktorá je postupnosťou

- veľkých písmen, malých písmen, číslic a podčiarnika, pričom začína malým písmenom (napr: *leto*, *dve_hodiny*, *rok2016*, *jeLuhar*),
- znakov medzi dvojicou apostrofov (napr: *'Q&A'*, *'dnes a zajtra'*),
- špeciálnych znakov (napr: +, =/=).

Premenné slúžia pre reprezentáciu objektov, ktoré v dobe tvorby programu nie je možné pomenovať. Sú to postupnosti veľkých písmen, malých písmen, číslic a podčiarnika, ktoré začínajú veľkým písmenom alebo podčiarnikom (napr: *X*, *Jano*, *_zima*). Samotný podčiarnik má špeciálny význam, reprezentuje *anonymnú* premennú.

Štruktúry sú zloženými termami, slúžia pre pomenovanie objektov s vnútornou štruktúrou. Skladajú sa z mena, ktoré je nasledované sekvenciou argumentov (ich počet sa označuje ako *árnosť*). Meno musí byť atómom, argument môže byť ľubovoľným termom. Príkladom sú štruktúry: *pozna(jano,anka)*, *f(X)*, *nema_rad(Kto,pitie(Co))*. Pre zoznamy sa používa špeciálna syntax – prvky zoznamu sú uzavreté v hranatých zátvorkách a navzájom oddelené čiarkou (napr: *[1,dva,Vela]*).

Prologovský program pozostáva z *klauzúl*, ktoré sú dvojakého typu: *fakty* a *pravidlá*. Každá klauzula je ukončená bodkou a medzerou alebo znakom nového riadku. Fakty sú štruktúry ľubovoľnej árnosti, ktoré definujú to, čo platí. Pravidlá reprezentujú vzťah medzi hlavou pravidla a telom pravidla. Pravidlo v predikátovom počte

$$osoba(Kto) \wedge vyrok(Co) \wedge osoba(Komu) \rightarrow povedal(Kto, Co, Komu)$$

má v prologovskom zápise tvar zmenený na

$$povedal(Kto, Co, Komu) \ :- \ osoba(Kto), vyrok(Co), osoba(Komu).$$

Dochádza k prehodeniu strán pravidla a zámene znaku implikácie za špeciálny atóm. Čiarka ako oddeľovač reprezentuje konjunkciu, disjunkciu možno reprezentovať bodkočiarkou.

```
pravda(je(poctivec,poctivec)).
pravda(je(luhar,luhar)).

klamstvo(je(poctivec,luhar)).
klamstvo(je(luhar,poctivec)).

ostrovan(jano,poctivec).
ostrovan(juro,poctivec).
ostrovan(fero,luhar).

povedal(Meno,poctivec,je(Kto,Kmen,Co)) :-
    ostrovan(Meno,poctivec),
    ostrovan(Kto,Kmen),
    pravda(je(Kmen,Co)).
povedal(Meno,luhar,je(Kto,Kmen,Co)) :-
    ostrovan(Meno,luhar),
    ostrovan(Kto,Kmen),
    klamstvo(je(Kmen,Co)).
```

Obr. 3.3: Kód v jazyku Prolog

logických viet v tvare definitných klauzúl, ktoré reprezentujú fakty a pravidlá platné v nejakej vybranej doméne. S jazykom sa pracuje interaktívnym spôsobom – je mu možné klásť otázky, ktoré sa snaží zodpovedať na základe svojej databázy. Otázka je považovaná za výrok, ktorý sa snaží na základe obsahu databázy dokázať, resp. určiť podmienky, pri platnosti ktorých tento výrok platí (vo forme potrebných unifikácií premenných) – systém má k dispozícii odvodzovací mechanizmus, využívajúci spätné reťazenie pravidiel.

Syntax jazyka je podobná syntaxi predikátového počtu s niekoľkými malými zmenami, takže prepis pravidiel vyjadrených pomocou predikátového počtu do prologovskej syntaxe je pomerne priamočiary. Ako ukážku možno uviesť reprezentáciu úlohy č. 2 z ilustračného príkladu Pr. 3.1. Výsledný prologovský program je na Obr. 3.3 – obsahuje sedem faktov a dve pravidlá, definujúce doménu ostrova poctivcov a luhárov.

Interakcia s Prologom

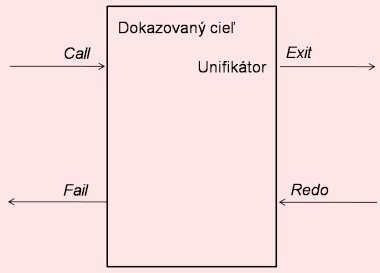
Pre vyriešenie úlohy je po zavedení programu do systému potrebné systému položiť otázku, ktorá v našom prípade reprezentuje výroky konkrétnych obyvateľov ostrova (čo povedal A a čo povedal B v úlohe 2 v príklade Pr. 3.1)

```
?- povedal(Am,A,je(Bm,B,luhar)), povedal(Bm,B,je(Am,A,C)).
```

kde atóm ?- reprezentuje systémovú nápoveď, indikujúcu očakávanie otázky. Keďže danú konjunkciu je možné dokázať, ako odpoveď systém poskytne informáciu o unifikácii všetkých piatich premenných. Pri takejto forme otázky odpoveď bude mať tvar

```
Am = jano,
A = poctivec,
Bm = fero,
B = C, C = luhar    ;
...
Am = fero,
A = C, C = luhar,
Bm = juro,
B = poctivec        ;
false
```

kde najprv sa zobrazia prvé štyri riadky (prvé riešenie). Znak bodkočiarky bol zadaný užívateľom – užívateľ pomocou tohto znaku môže požadovať nájdenie alternatívneho riešenia[6]. Posledný riadok indikuje, že ďalšie alternatívne riešenie už neexistuje. Ak by namiesto bodkočiarky užívateľ zadal nový riadok, hľadanie sa ukončí.

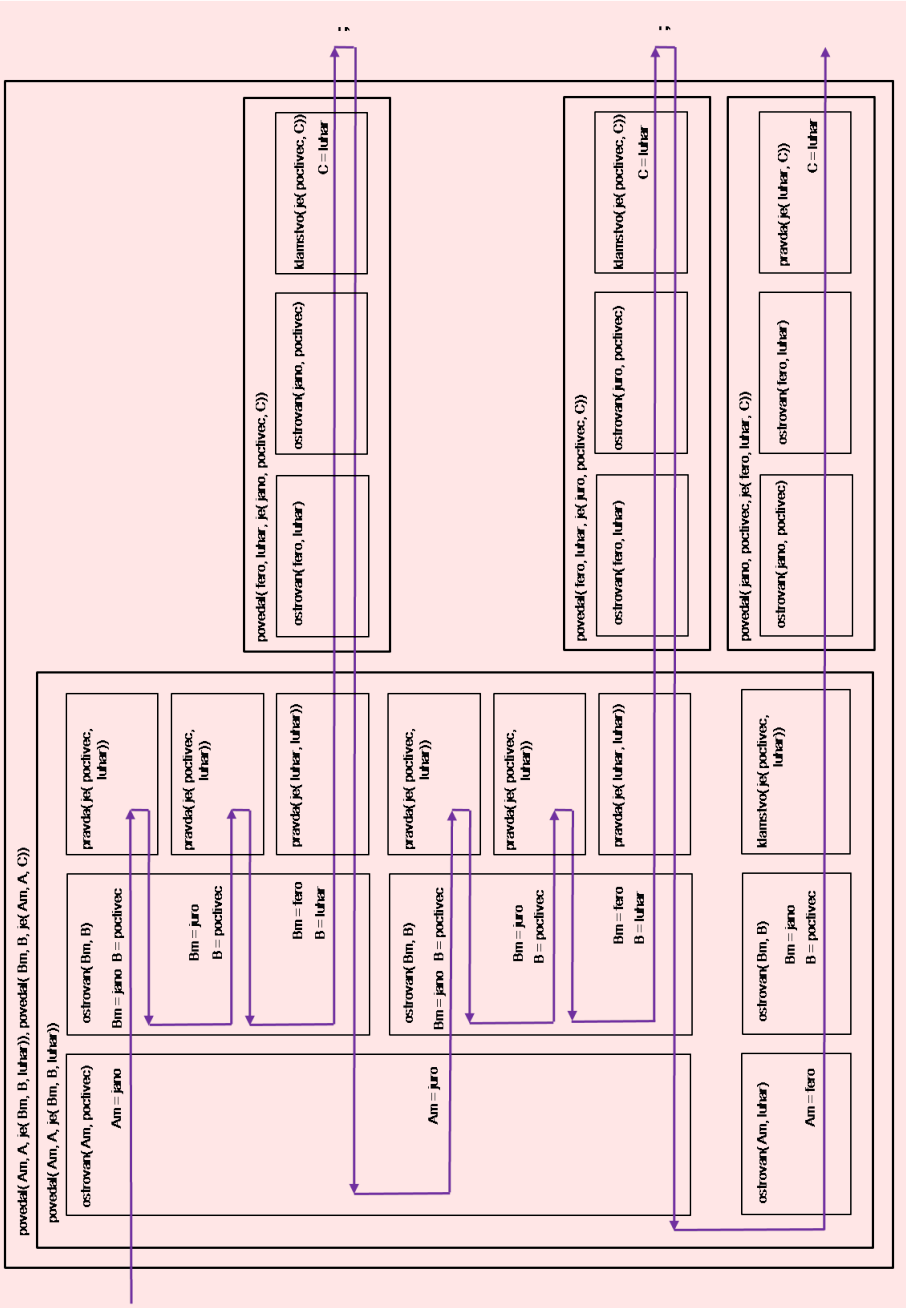


Obr. 3.4: Blok blokového modelu

BLOKOVÝ MODEL HĽADANIA RIEŠENIA

Proces hľadania riešenia je možné vizualizovať pomocou blokového modelu. V tomto type modelu každému cieľu zodpovedá jeden blok, pričom cieľ je explicitne uvedený v hornej časti bloku (ukážka bloku je na Obr. 3.4). Vnoreniu blokov zodpovedá vnorenie cieľov, keď plnenie nejakého cieľa, unifikovateľného s hlavou nejakého pravidla (reprezentovaného vonkajším blokom), je nahradené plnením cieľov v tele tohto pravidla (reprezentovaných vloženými vnútornými blokmi). Samotný proces hľadania je reprezentovaný

[6]Bodkočiarka sa používa v CLI implementáciách dialógu so systémom Prolog. Pri použití grafického rozhrania môže byť žiadosť inicializovaná nejakým interaktívnym prvkom.



Obr. 3.5: Blokový model hľadania riešenia

orientovanou čiarou, prechádzajúcou jednotlivými blokmi podľa toho, ako postupuje plnenie zodpovedajúcich cieľov.

Brány modelu

Každý blok má v princípe štyri brány – dve na pravej a dve na ľavej strane. Na ľavej strane sa nachádzajú brány *Call* (vstupná brána) a *Fail* (výstupná brána), na pravej strane to sú brány *Exit* (výstupná brána) a *Redo* (vstupná brána). Vstup cez Call bránu znamená, že začína pokus o splnenie cieľa, o jeho unifikáciu s nejakým faktom alebo hlavou niektorého z pravidiel. Ak pokus bol neúspešný, nasleduje výstup z daného bloku cez Fail bránu. Ak však bol pokus úspešný a bola nájdená vhodná klauzula, tak nasleduje výstup cez bránu Exit, pričom prípadný unifikátor je uvedený pri danej bráne. V prípade, že sa neskôr ukáže, že nájdený spôsob splnenia daného cieľa nebol vhodný a je potrebné hľadať jeho alternatívne splnenie, opätovne sa vstupuje do bloku bránou Redo.

V prípade úspešnej unifikácia a výstupu cez bránu Exit sú v diagrame uvedené aj viazania premenných, ktoré sa vyskytujú v spĺňanom cieli, spôsobené unifikáciou.

Blokový model zodpovedajúci nášmu ilustračnému príkladu je zobrazený na Obr. 3.5. Samotný model reprezentuje spĺňanie konjunkcie dvoch cieľov a preto bloky, reprezentujúce tieto ciele, sú vnorené v bloku, reprezentujúcom danú konjunkciu. A keďže každý z týchto cieľov je spĺňaný pomocou pravidla, ktorého telo obsahuje tri podciele, dochádza k ďalšiemu vnáraniu blokov.

Tak pre príklad na obrázku pri spĺňaní prvej časti otázky pomocou tela prvého pravidla došlo po splnení prvého podcieľa `ostrovan(Am,poctivec)` (brána Exit s *Am=jano*) a druhého podcieľa `ostrovan(Bm,B)` (cez bránu Exit s *Bm=jano* a *B=poctivec*) k pokusu o splnenie tretieho podcieľa pravidla `pravda(je(poctivec,luhar)` (cez bránu Call) – avšak neúspešne (návrat zo spĺňania tohto tretieho podcieľa cez bránu Fail). Nasledovala snaha o alternatívne splnenie druhého podcieľa `ostrovan(Bm,B)` (opätovný návrat do spĺňania druhého podcieľa cez bránu Redo). Tento pokus bol síce úspešný (tentoraz Exit s *Bm=juro* a *B=poctivec*), avšak výsledok snahy o splnenie tretieho podcieľa pravidla dopadol rovnako neúspešne ako v predchádzajúcom prípade (ďalší návrat do spĺňania druhého podcieľa cez bránu Redo). Až tretí pokus o splnenie druhého podcieľa bol úspešný (Exit s *Bm=fero* a *B=luhar*) s následným úspechom na treťom podcieli (brána Exit) a tým pádom aj pri spĺňaní nadradeného bloku (ešte jedna brána Exit). Splnenie druhého cieľa konjunkcie už bolo priamočiare (kombinujúce prechody cez brány Call a Exit). Až používateľ vyvolal snahu o alternatívu, ktorá začína sekvenciou troch Redo brán.

Rozšírenie: Vybrané vlastnosti jazyka Prolog

Ukážka vlastností Prologu

Pre ilustráciu niektorých vlastností jazyka Prolog, ktoré ho odlišujú ako od predikátovej logiky tak aj od iných programovacích jazykov, uvažujme krátky program, slúžiaci k potvrdeniu prítomnosti nejakého prvku v zozname.

```
member(X,[X|_]).
member(X,[_|Y]) :- member(X,Y).
```

Program je jednoduchý – nejaký prvok sa vtedy nachádza v zozname, ak je v tom zozname na prvej pozícii alebo na nejakej inej pozícii. Zápis $[A|B]$, použitý v programe, reprezentuje zoznam ktorý má na začiatku nejaký prvok A a ktorého zvyšok je B (čo môžu byť ďalšie prvky alebo ten zvyšok môže byť už prázdny zoznam).

Tento program môže byť použitý napríklad v súvislosti s dokazovaním platnosti cieľa $member(3, [1, dva, 3])$. Pokus o dokázanie pravdivosti tohto cieľa je úspešný. Ak by sa však použil cieľ $member(jeden, [1, dva, 3])$, tak takýto cieľ sa na základe programu reprezentujúceho znalostnú bázu dokázať nedá – a nedá sa ani dokázať jeho neplatnosť. Na rozdiel od predikátovej logiky však Prolog o tomto cieli bude tvrdiť, že je nepravdivý. To preto, pretože na rozdiel od logiky používa princíp *uzavretého sveta*: to, čo nie je explicitne definované ako platné resp. čoho platnosť sa nedá dokázať, je považované za neplatné.

Na rozdiel od iných programovacích jazykov, v Prologu nie je pevne stanovené, ktorý argument je vstupný a ktorý výstupný. Preto sa daný program dá použiť aj ako generátor

```
?- member(X,[1,dva,3]).
X = 1                   ;
X = dva                 ;
X = 3                   ;
false.
```

ktorý postupne zdeľuje, ktoré prvky sú obsiahnuté v zadanom zozname. Taktiež ho možno použiť pre vysvetlenie, ako môže vyzerať zoznam, ktorý obsahuje nejaký zadaný prvok

```
?- member(3,X).
X = [3|_G268]               ;
X = [_G267, 3|_G271]        ;
X = [_G267, _G270, 3|_G274] ;
```

pričom analogicky možno pokračovať až do nekonečna alebo do zlyhania kvôli nedostatku pamäti (podľa toho čo nastane skôr :-).

Rozšírenie: Vybrané vlastnosti jazyka Prolog (pokr.)

Zameniteľnosť vstupných a výstupných argumentov však neplatí univerzálne. Pre zvýšenie efektívnosti zabudované literály sú "dokazované" vykonaním im prislúchajúceho kódu namiesto skutočného dokazovania. Tak napríklad je možné určiť súčet dvoch čísel X *is* $5+3$ alebo otestovať či tento súčet je rovný zadanej hodnote 9 *is* $5+3$, ale nie je možné rozdeliť číslo na dva sčítance ani v tom prípade, keď by toto rozdelenie bolo jednoznačné ako napríklad 9 *is* $X+5$.

Podobne ako v predchádzajúcom prípade by k zlyhaniu došlo aj v tej situácii, keď by v definícii predikátu *member* boli klauzuly uvedené v opačnom poradí, najprv pravidlo a až potom fakt. Na rozdiel od predikátovej logiky by sa Prolog dostal do nekonečnej rekurzie (napr. pri dokazovaní cieľa `member(3,X)`), pretože poradie klauzúl v programe má vďaka prehľadávaniu do hĺbky rozhodujúci význam.

Na rozdiel od predikátovej logiky, Prolog má prostriedky pre modifikáciu svojej znalostnej bázy – pomocou *retract* vie odstraňovať klauzuly a pomocou *asserta* alebo *assertz* dokáže zase klauzuly pridávať (na začiatok alebo na koniec). Jednoduchou ukážkou manipulácie s faktami je program pre súčet zoznamu čísel

```
sucet(Z,_) :- member(X,Z), asserta(cislo(X)), fail.
sucet(_,_) :- assertz(vysl(0)), fail.
sucet(_,_) :- retract(cislo(X)), retract(vysl(Y)),
              Y1 is Y + X, assertz(vysl(Y1)), fail.
sucet(_,X) :- retract(vysl(X)).
```

ktorý túto manipuláciu kombinuje s jedným z predikátov ovplyvňujúcich proces prehľadávania. Je to zabudovaný predikát *fail*, ktorý je vždy nesplneným cieľom (a teda podporujúcim iba brány Call a Fail).

Cvičenia

1. Reprezentáciu problému programom podľa Obr. 3.1 vyskúšajte v systéme Clips. S využitím jeho nástrojov sledujte zmeny faktov v databáze faktov a agendy počas krokovania behu programu.

2. Navrhnite zásadu pre efektívne usporiadanie podmienok v LHS časti pravidiel. Pri úvahách použite dve rôzne usporiadania podmienok

(defrule usporiadanie_1	(defrule usporiadanie_2
(podmienka ?a ?b ?c)	(cislo ?a)

```
(cislo ?a)                  |   (cislo ?b)
(cislo ?b)                  |   (cislo ?c)
(cislo ?c)                  |   (podmienka ?a ?b ?c)
=> ... )                    |   => ... )
```

pričom uvažujte prítomnosť nasledovných faktov

```
(podmienka 1 3 5)   (podmienka 2 4 6)   (podmienka 3 5 7)
(cislo 1)   (cislo 2)   (cislo 3)   (cislo 4)   (cislo 5)
(cislo 6)   (cislo 7)   (cislo 8)   (cislo 9)   (cislo 10)
```

3. Negácia vzoru je v Reteho sieti modelovaná *negačným* uzlom s dvomi vstupmi a jedným výstupom. Pravý vstup je napojený na pamäťový uzol, reprezentujúci negovaný vzor. Negačný uzol určuje pre každý prvok v pamäti, na ktorú je pripojený ľavý vstup, počet prvkov v pravej pamäti, s ktorými je konzistentný. Na základe týchto počtov sú potom prvky z ľavého vstupu presúvané na výstup – presúvané sú iba tie prvky, ktorým zodpovedá nulový počet (a teda nie sú konzistentné so vzorom, ktorý má byť negovaný). Ukážte, že pri takejto reprezentácii negácie:

   (a) v podmienkach pravidiel nemôžu byť iba negované vzory,
   (b) neplatí ekvivalencia medzi nejakým vzorom a dvojitou negáciou tohto vzoru.

4. Vytvorte pravidlá pre zistenie, či na ostrove žije viac poctivcov alebo luhárov. Použite rovnaké fakty ako je uvedené na Obr. 3.1. Vytvorte Reteho sieť a simulujte propagáciu faktov touto sieťou, ak fakty boli vložené pomocou konštruktu *deffacts*. Riešenie založte na:

   (a) párovom porovnávaní luhárov a poctivcov,
   (b) určení počtu luhárov a počtu poctivcov.

5. Namodelujte pomocou produkčných pravidiel zložitejšiu križovatku riadenú pomocou dopravných značiek. Vytvorené pravidlá odskúšajte v systéme Clips pre rôzne dopravné situácie.

6. Vyhľadajte popis vernostného systému niektorej z leteckých spoločností. Analyzujte tento popis a vytvorte jeho model v tvare produkčných pravidiel. Overte jeho činnosť v systéme Clips.

7. Manuálne simulujte vykonávanie algoritmu Alg. 3.3 pre dôkaz platnosti cieľa $j(X)$ pri použití znalostnej bázy:

$$\begin{array}{rl} a(X) \wedge b(X) \rightarrow e(X) & c(bolo) \wedge e(X) \rightarrow f(bolo, X) \\ e(X) \wedge d(bude) \rightarrow f(bude, X) & c(je(X)) \wedge d(je(X)) \rightarrow g(je, X) \\ i(Y) \wedge f(X, Y) \rightarrow j(X) & i(X) \wedge g(Y, X) \rightarrow j(Y) \\ a(dnes) \quad b(dnes) & c(A) \quad d(B) \quad i(dnes) \end{array}$$

   Následne prepíšte do tvaru kódu v jazyku Prolog a overte v systéme Prolog.

8. Reprezentáciu problému programom podľa Obr. 3.3 vyskúšajte v systéme Prolog. S využitím jeho trasovacích nástrojov sledujte následnosť pokusov o splnenie resp. opakované splnenie cieľov. Dokreslite blokový diagram na Obr. 3.5 pre prípad ďalšieho zadávania bodkočiarky.

9. Program z predchádzajúcej úlohy rozšírte tak, aby vo výrokoch obyvateľov ostrova sa mohla vyskytovať aj konjunkcia a disjunkcia. Následne riešte rozhovory (z ktorých každý je monológom jedného obyvateľa):

   (a) A povedal: "Jeden z A a B je luhár."

   (b) A povedal: "B je poctivec alebo ja som luhár."

   (c) A povedal: "Som luhár ale B je poctivec."

10. Program z úlohy na Obr. 3.3 rozšírte tak, aby vo výrokoch obyvateľov bolo možné zohľadniť tieto rozhovory:

   (a) - A povedal: "Všetci z nás sú luhári."
   - B povedal: "Jeden z A, B a C je poctivec."

   (b) - A povedal: "Všetci z nás sú luhári."
   - B povedal: "Jeden z A, B a C je luhár."

   (c) - A povedal: "Nik z nás nie je luhár."
   - B povedal: "Jeden z nás je luhár."
   - C povedal: "Dvaja z nás sú luhári."
   - D povedal: "Traja z nás sú luhári."

11. Program z predchádzajúcej úlohy rozšírte o možnosť jednej metaúrovne (aby bolo možné pracovať s výrokmi o výrokoch):

   (a) - B povedal: "A povedal o sebe že je luhár."

- C povedal: “B je luhár.”

(b) - B povedal: “A povedal o A, B a C, že iba jeden z nich je poctivec.”
- C povedal: “B je luhár.”

Matematické symboly

Symbol	Krátky popis
$. \mid .$	alternatívy
$::=$	definícia
$\bigvee_{i\ldots}$	disjunkcia viacerých symbolov
$\mathcal{D}$	doména
$\equiv$	ekvivalentnosť viet
$\exists$	existenčný kvantifikátor
$I(.)$, $.^I$	interpretácia symbolu, predikátu
$\begin{pmatrix} . \\ . \end{pmatrix}$	kombinačné číslo
$\bigwedge_{i\ldots}$	konjunkcia viacerých symbolov
$\models$	logické vyplývanie
LHS/RHS	ľavá/pravá strana pravidla
$\mid . \mid$	mohutnosť (počet prvkov) množiny
$\leq_k$, $\geq_k$, $=_k$	najviac k, najmenej k, presne k
$\vee$	operátor disjunkcie
$\leftrightarrow$	operátor ekvivalencie
$\rightarrow$	operátor implikácie
$\wedge$	operátor konjunkcie
$\neg$	operátor negácie
$\#$	počet
$\subseteq$	podmnožina

Symbol	Krátky popis
$< . >+$	povinný viacnásobný výskyt (1 alebo viac)
$\square$	prázdna klauzula, spor, neúspech
$[\,]$	prázdny zoznam
$:=$	priradenie hodnoty
$\in$	prvok množiny
$\setminus$	rozdiel množín
$\prod_{i=1,\ldots}$	súčin
$\bot$	symbol, ktorý je vždy nepravdivý
$\top$	symbol, ktorý je vždy pravdivý
P, Q, R, S	symboly, literály
θ	unifikátor
$\forall$	univerzálny zovšeobecňovací kvantifikátor
$\| . \|$	veľkosť logickej vety
$< . >^*$	voliteľný viacnásobný výskyt (0 alebo viac)
$\lceil . \rceil$	zaokrúhlenie nahor
$\cup$	zjednotenie
KB	znalostná báza

Literatúra

[1] Bench-Capon, T.J.M.: Knowledge representation: An Approach to Artificial Intelligence. Academic Press, 2014.

[2] Biere, A. a kol.: Handbook of Satisfiability. IOS Press, Amsterdam, 2009.

[3] Brachman, R.J. – Levesque, H.J.: Knowledge Representation and Reasoning. Morgan Kaufmann, San Francisco, CA, 2004.

[4] Coppin, B.: Artificial Intelligence Illuminated. Jones and Bartlett Publishers, Sudbury, MA, 2004.

[5] Csontó, J. – Sabol, T.: Umelá inteligencia. Edičné stredisko TU, Košice, 1991.

[6] Flach, P.: Simply Logical: Intelligent reasoning by example. John Wiley & Sons, Chichester/New York, 1994.

[7] Gabbay, D.: Logic for Artificial Intelligence and Information Technology. College Publications, London, 2007.

[8] Gelfond, M. – Kahl, Y.G.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents. Cambridge University Press, New York, NY, 2014.

[9] Holldobler, S. – Nguyen, V.H.: An Efficient Encoding of the at-most-one Constraint. KRR Report 13-04, Technische Universität Dresden, Institute of Artificial Intelligence, 2004.

[10] Kostelník, P.: Praktický úvod do symbolickej umelej inteligencie. In: Umelá inteligencia a kognitívna veda I, Kvasnička, V. et al. (eds.), STU, Bratislava, 2009, 87–138.

[11] Košík, M.: Prirodzená dedukcia. In: Umelá inteligencia a kognitívna veda II, Kvasnička, V. et al. (eds.), STU, Bratislava, 2010, 125–146.

[12] Kroening, D. – Strichman, O.: Decision Procedures: An Algorithmic Point of View. Springer, 2008.

[13] Návrat, P. a kol.: Umelá inteligencia. Slovenská Technická Univerzita, Bratislava, 2002.

[14] Nieuwenhuis, R. – Oliveras, A. – Tinelli, C.: Solving SAT and SAT Modulo Theories: From an Abstract Davis-Putnam-Logemann-Loveland Procedure to DPLL(T). Journal of the ACM, 53 (6), 2006, 937–977.

[15] Petke, J.: Bridging Constraint Satisfaction and Boolean Satisfiability. Springer, Cham, 2015.

[16] Poole, D. – Mackworth, A.: Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press, 2010.

[17] Russel, S. – Norvig, P.: Artificial Intelligence: A Modern Approach (3. vydanie). Pearson, Upper Saddle River, NJ, 2010.

[18] Shapiro, S.C.: Knowledge Representation. In Lynn Nadel, Ed., Encyclopedia of Cognitive Science, Volume 2, Macmillan Publishers Ltd., 2003, 671-680.

[19] van Harmelen, F. – Lifschitz, V. – Porter, B.: Handbook of Knowledge Representation. Elsevier, Amsterdam, 2008.

Táto publikácia bola vytvorená za podpory projektu KEGA č. 034TUKE-4/2014 "Integrácia študijných programov v odboroch Kybernetika a Umelá inteligencia".

Autor:	Mach Marián
Názov:	Reprezentácia znalostí a riešenie úloh
Podnázov:	Logické prístupy
Recenzenti:	doc. Ing. Kristína Machová, PhD., Ing. Tomáš Cádrik, Ing. Martin Čertický
Vydanie:	prvé
Náklad:	60 ks
Rozsah:	96 strán
Vydavateľ:	Technická univerzita v Košiciach
Sadzba:	LaTeX
Formát:	b5
Rok:	2016

Rukopis neprešiel jazykovou úpravou. Za odbornú a obsahovú stránku zodpovedá autor.

ISBN 978-80-553-2632-0