Reprezentácia znalostí prostriedkami logického programovania

Ján Šefránek

Ústav informatiky, Fakulta matematiky, fyziky a informatiky Univerzity Komenského, Bratislava, e-mail: sefranek@fmph.uniba.sk

Abstrakt O reprezentácii znalostí sa veľmi často rozmýšľa na pomerne nízkej úrovni (rámce, siete, pravidlá atď.). Ide v podstate o úroveň dátových štruktúr, s ktorými manipulujú nejaké procedúry. Inferencia sa vtedy chápe čisto procedurálne. Jedným z cieľov príspevku je poukázať na slabiny takéhoto prístupu. Hlavným cieľom však bude predstaviť teóriu reprezentácie znalostí, založenú na rôzne silných jazykoch logického programovani.

1 Úvod

Výskum v oblasti reprezentácie znalostí prešiel minimálne troma vývojovými fázami. V počiatočnom období umelej inteligencie (päťdesiate a šesťdesiate roky) dominovala predstava, že stačí opísať nejakú doménu pomocou formúl prvorádovej logiky a pomocou automatického dokazovania získať dôsledky, potrebné pre riešenie daného problému. Skúsení logici videli hneď v počiatkoch úskalia tohto programu.[1] Ostatní museli na základe skúsenosti zistiť, že bariéra nevypočítateľnosti, a principiálna neúplnosť našich poznatkov odsudzujú takýto program na neúspech.

Jedným z najznámejších kritikov programu bol Minski [16], ktorý v opozícii k dôrazu na dedukciu prišiel s ideou nemonotónnosti usudzovania a zdôrazňoval zhlukovanie poznatkov do celkov, ktoré nazval *rámcami*. Neskorší výskum a implementácie však ideu rámca zosunuli do úrovne údajových štruktúr a k odvodzovaniu sa pristupovalo procedurálne. Spravidla absentovala sémantická charakterizácia procedúr, manipulujúcich s reprezentovanými poznatkami. Navyše, reprezentácia poznatkov a odvodzovanie sa chápali ako odlišné a oddelené domény a schopnosti. (Kvôli zjednodušeniu vyjadrovania budeme o tomto prístupe neskôr hovoriť ako o *klasickom prístupe* k reprezentácii znalostí.)

Nemonotónne usudzovanie sa stalo predmetom záujmu teoretikov už koncom sedemdesiatych rokov [3]. Postupne sa ukázala tesná súvislosť tohto výskumu s výskumom v logickom programovaní a deduktívnych databázach [19]. Význam výskumu v oblasti nemonotónneho usudzovania a logického programovania pre reprezentáciu poznatkov sa zdôrazňuje už od začiatku deväťdesiatych rokov [4]. Dôraz sa kládol na fundamentálne problémy reprezentácie znalostí - sémantiku,

[1] Pozri Bar Hillelov komentár k McCarthyho vízii, publikovaný v [15].

vyjadrovaciu silu a výpočtovú zložitosť. Vytvoril sa jednotiaci pohľad na bázu poznatkov a odovodzovanie. Položili sa pevné sémantické základy hypotetickému usudzovaniu, ktoré je kľúčové z hľadiska spracovania znalostí. Od polovice deväťdesiatych rokov sa veľká pozornosť venuje implemenáciám [17, 8, 14, 18, 5]. O tomto prístupe budeme ďalej hovoriť ako o prístupe, založenom na logickom programovaní.

Problém. Medzi klasickým prístupom a prístupom, založenom na logickom programovaní možno vidieť latentné napätie. Ide predovšetkým o napätie medzi zdôrazňovaním a zanedbávaním sémantických základov znalostných systémov. Prirodzeným dôsledkom zdôrazňovania (zanedbávania) sémantiky je jednotiaci (ostro oddeľujúci) pohľad na reprezentáciu a odvodzovanie. Oba prístupy existujú popri sebe, ale bez vzájomného ovplyvňovania. Najmä v našom prostredí je prístup, založený na logickom programovaní takmer neznámy.

Cieľ článku. Článok je prehľadový. V istom zmysle slova je aj osvetový. Jeho cieľom je informovať o relevancii výskumu v logickom programovaní a nemonotónnom usudzovaní pre pochopenie reprezentácie poznatkov. Oveľa podrobnejšie o téme pojednáva [20]. Domnievame sa, že v našom prostredí je takýto prehľad nanajvýš potrebný. Motivácia pre jeho napísanie sa vynorila počas komunikácie s popredným predstaviteľom komunity, sústredenej okolo série konferencií Znalosti.

Stručný obsah. V časti 2 budeme argumentovať, prečo je užitočné a adekvátne považovať usudzovanie[2] za integrálnu súčasť reprezentácie znalostí. Hlavný argument bude sémantický.

Preto v časti 3 budeme venovať pozornosť sémantickým otázkam. Poukážeme na absenciu sémantických základov pri tradičnom prístupe k reprezentácii znalostí a schématicky objasníme dva základné prístupy k sémantickej charakterizácii (nemonotónneho) usudzovania. Použitím jedného z nich dostávame sémantiku stabilných modelov. Jej zovšeobecnením sú sémantiky odpoveďových množín pre rôzne silné jazyky logického programovania (resp. answer set programming).[3]

V časti 4 načrtneme, čo sa dá vyjadriť, keď príjmeme, že rôzne jazyky logického programovania považujeme za jazyky na reprezentáciu znalostí (so sémantikou odpoveďových množín v pozadí) a akú pridanú hodnotu získame. Na záver komentáre a diskusia.

---

[2] Budeme používať termín *usudzovanie*, ktorý možno znie v spojení s programami alebo bázami znalostí neprirodzene. Držíme sa jednak zaužívaného anglického termínu *reasoning*, jednak pripomíname, že nám ide o formalizáciu a výpočtovú realizáciu spôsobov, ako ľudia usudzujú, keď spracúvajú znalosti. Nebudeme sa však vyhýbať takým termínom ako inferencia alebo odvodenie. Budeme ich považovať za synonymá, používame ich však iba pri neformálnom opise.

[3] Zdôrazňujeme, že tradičná reprezentácia pomocou pravidiel bez explicitnej a detailnej sémantiky – v kontraste k pravidlám logického programovania – je vo svojej podstate tiež reprezentáciou na úrovni údajových štruktúr, nad ktorou operujú nejaké procedúry.

2 Usudzovanie ako kľúč k reprezentácii znalostí

Tradične sa pomerne ostro rozlišuje medzi reprezentáciou znalostí na jednej strane a usudzovaním na strane druhej. Takáto ostrá hranica však nie je dobre zdôvodnená.

Zaujímajú nás systémy, ktoré používajú pri realizácii výpočtových úloh znalosti. Nie je dôležité, či znalosť, potrebnú na vykonanie nejakej úlohy (zodpovedanie nejakej otázky) systém získa prehľadávaním bázy znalostí alebo, či ju odvodí. Prehľadávanie bázy znalostí je iba triviálne odvodenie (počiatočná fáza procesu usudzovania). Navyše, očakáva sa, že riešenie úloh s využitím znalosti má byť do značnej miery flexibilné, schopné zvládať výnimky, neobvyklé situácie, nedostatok potrebných informácií. Inými slovami: má využívať hypotetické usudzovanie.

Pre tento typ úloh je výhodný deklaratívny jazyk na vyjadrenie, zápis (reprezentáciu) znalostí a na formulovanie požiadaviek. Štúdiu a charakterizácii takýchto jazykov sa venuje logika. Úloha logiky pre pochopenie inteligencie, založenej na spracovaní znalostí sa často prirovnáva k úlohe matematickej analýzy pre fyziku.

Budeme ďalej predpokladať, že výpočtové systémy nebudú mať všetky potrebné znalosti explicitne zapísané v nejakej databáze (báze znalostí), ale budú schopné odvodiť si aj dôsledky toho, čo je explicitne zapísané v báze znalostí, prípadne vygenerovať hypotézy, zlučiteľné s obsahom bázy znalostí.

Predstavu, ktorá nás vedie k tomu, že nie je dôležité rozlišovať medzi znalosťami reprezentovanými explicitne a znalosťami získanými usudzovaním, zhrnieme takto:

Znalosti, ktoré má agent k dispozícii, sú vyjadrené (reprezentované) v nejakom jazyku J. Nejaká množina E viet (formúl) jazyka J reprezentuje tie znalosti, ktoré sú explicitne zapísané (v báze). Znalosti, získané usudzovaním, sú opäť vety jazyka J. Vety, ktoré agent odvodí z E označíme ako $Cn(E)$. Pritom Cn pozažujeme za *operátor odvodenia*, ktorý množinám viet priraďuje množiny viet. Je pritom definovaná nejaká *sémantika* σ, ktorá výrazom jazyka priraďuje nejaké objekty. *Sémantickou špecifikáciou* operátora odvodenia Cn možno nazvať špecifikáciu nejakého vzťahu medzi $\sigma(E)$ a $\sigma(Cn(E))$ pre ľubovoľné E.

Obvykle sa $\sigma(E)$ definuje ako nejaká trieda modelov. Ak teda $\sigma(E)$ je nejaká trieda modelov množiny E, potom sa $Cn(E)$ zvykne definovať ako množina všetkých viet pravdivých v $\sigma(E)$, teda $\sigma(E) = \sigma(Cn(E))$.[4] Uvedenou šablónou sa pozrime na dedukciu: *Dedukcia* (z nejakej teórie, či bázy poznatkov) sa charakterizuje pomocou triedy *všetkých* jej modelov. Keď $\sigma(E)$ je trieda všetkých modelov bázy E, potom množina deduktívnych dôsledkov množiny E, označme ju $Th(E)$, je množinou všetkých viet pravdivých v každom modeli zo $\sigma(E)$. Neskôr sa budeme venovať sémantickej špecifikácii ne-deduktívneho oddvodenia.

[4] Treba tu iba poznamenať, že operátory odvodenia, zaujímavé z hľadiska reprezentácie znalostí majú často iné vlastnosti. Zaujímavou vlastnoťou je alternatívnosť, pozri napríklad [20].

Môžeme teda uzavrieť: Ak si zvolíme nejakú sémantiku explicitne reprezentovaných poznatkov, potom tá si vynucuje, akú množinu poznatkov musíme akceptovať. Inak nami definovanú sémantiku neberieme vážne. Možno treba ešte venovať pozornosť otázke, či vôbec treba nejakú sémantiku.

3 Sémantika a reprezentácia poznatkov

Najprv rozlíšime dva základné typy sémantickej charakterizácie hypotetického (nemonotónneho)[5] usudzovania, ktoré má kľúčovú úlohu z hľadiska reprezentácie poznatkov. Treba však pripomenúť, že ide o schematický pohľad na množstvo rôznych prístupov. Potom ukážeme niekoľko príkladov, demoštrujúcich nedostatočnosť takých prístupov k reprezentácii poznatkov a inferencii, ktoré nie sú založené na nejakej sémantike.

3.1 Sémantiky nemonotónneho usudzovania

Už dávno sa v teórii a praxi znalostných systémov akceptuje, že realistický prístup k problému reprezentácie poznatkov spočíva na uvedomení si neúplnosti poznania, ktoré je k dispozícii. Preto sa považuje za nutné pracovať s predbežnými závermi, odvolateľnými hypotézami, revidovať bázy poznatkov, vrátane poznatkov, ktoré sú reprezentované implicitne. To znamená, že potrebujeme usudzovanie, ktoré je nemonotónne: po pridaní nových poznatkov do bázy sme niekedy nútení vzdať sa niektorých predbežných záverov, ak nie sú zlučiteľné s novými poznatkami.

Možno rozlišovať dve triedy sémantík nemonotónneho usudzovania. Prvá z nich – nazvime ju *preferenčnou sémantikou* – vychádza z toho, že niektoré modely sú viac dôležité, iné menej. Formálne, definuje sa nejaká relácia preferencie $\prec$ na modeloch. M, model bázy E je maximálne preferovaný vtedy. keď neexistuje jej model N, ktorý by bol preferovanejší, $(\neg\exists N) N \prec M$. Typickým príkladom sú minimálne modely (ten, kto si pripravuje plán cestovania železnicou, pracuje s minimálnym modelom: za pravdivé považuje iba to a presne to, čo je napísané v cestovnom poriadku).

Keď $\sigma_\prec(E)$ je trieda všetkých maximálne preferovaných modelov bázy E, potom množina viet, ktoré preferenčne vyplývajú z množiny E, označme ju $Cn_\prec(E)$, je množinou všetkých viet pravdivých v každom modeli zo $\sigma(E)$.

Prejdime k druhej zo spomínaných tried sémantík, nazvime si ju *sémantikou pevných bodov*. Intuitívne ide o to, že sa konštruujú množiny tých hypotéz, ktoré dohromady dávajú zmysel, neodporujú si a ktoré vytvárajú čo najúplnejší obraz danej oblasti. Trochu viac technicky: Definuje sa operátor, ktorý množine viet priradí množinu viet. Tento operátor formalizuje dve dôležité črty - *nasýtenosť* ("výstupná" množina obsahuje všetky relevantné vety) a *koherenciu* (výstupné vety si vzájomne neodporujú). Pevné body takýchto operátorov sú význačnými

[5] Oba termíny považujeme v tomto texte za synonymné.

sémantickými objektami.[6] Keďže ide o komplikovanejšiu konštrukciu, uvedieme príklad - stabilné modely logických programov. Prezentovaný výklad z [21] sa líši od pôvodnej teórie Gelfonda a Lifschitza [11].

Predpokladáme nejakú množinu atómov $\mathcal{A}$ výrokovej logiky.[7] Literál je atóm $A \in \mathcal{A}$ alebo negácia atómu $notA$, kde $A \in \mathcal{A}$. Negácie atómov nazvime negatívnymi literálmi. V logickom programovaní sa, intuitívne, literál $notA$ považuje za pravdivý, ak nie je známe, že by bol pravdivý A. Teda $notA$ sa prijíma ako hypotéza dovtedy, kým nie je vyvrátená (pomocou A). Jedným z upresnení tejto intuície je práve sémantika stabilných modelov. Pravidlo je formula r tvaru $A \leftarrow A_1, \ldots, A_k, notB_1, \ldots, notB_m$, kde A, A_i, B_j sú atómy (pre každé $i = 1, \ldots, k$, $j = 1, \ldots, m$). Množinu literálov $\{A_1, \ldots, A_k, notB_1, \ldots, notB_m\}$ nazývame telom pravidla r (značenie $body(r)$) a atóm A nazývame hlavou pravidla ($head(r)$). Zamýšľaný význam pravidla: ak sú všetky literály $A_1, \ldots, A_k, notB_1, \ldots, notB_m$ pravdivé, potom je aj atóm A pravdivý.

Kripkeho štruktúra asociovaná s programom P je dvojica (W, ρ). W je množina konzistentných množín literálov, doplnená o symbol $w_\perp$, reprezentujúci každú nekonzistentnú množinu literálov. Relácia dostupnosti ρ je množina dvojíc (w_1, w_2) takých, že existuje pravidlo $r \in P$, pre ktoré $w_1 \models body(r)$ a $w_2 = w_1 \cup \{head(r)\}$, ak $w_1 \cup \{head(r)\}$ je konzistentná množina literálov. V opačnom prípade je $w_2 = w_\perp$. O prvkoch relácie dostupnosti budeme hovoriť ako o hranách a o ich postupnostiach ako o cestách.

Úplnú množinu literálov S (t.j. pre každý atóm $A \in \mathcal{A}$ buď $A \in S$ alebo $not\ A \in S$) voláme *stabilným modelom* výrokovologického programu P práve vtedy, keď je splnená nasledujúca podmienka: v Kripkeho štruktúre asociovanej s P existuje cesta, ktorej koreňom je S^-, množina všetkých negatívnych literálov z S, Táto cesta je ukončená v S, t.j. neexistuje hrana $(S, w_\perp)$. Idea: Ako hypotézy predpokladáme všetky negatívne literály z S^-. Pomocou programu P z nich postupne odvodzujeme všetky možné dôsledky. Ak dosiahneme tak nasýtenú množinu literálov, že z nich už nemožno pomocou P odvodiť nič nového, ani naraziť na nekonzistentosť, potom S možno považovať za význam programu P. S je pevným bodom operátora, ktorý interpretáciám priraďuje interpretácie (využívajúc pritom pravidlá programu P). Inými slovami: ak P považujeme za návod, ako meniť obraz o možnom stave sveta (reprezentovaný množinou literálov, pravdivých v tom stave), potom možný obraz sveta S je stabilným modelom P vtedy, keď P nedokáže transformovať S do inej množiny literálov (do iného obrazu sveta).

[6] Množina hypotéz, ktorá je pevným bodom takéhoto operátora, spĺňa predstavu o nasýtenej a koherentnej množine hypotéz. Nasýtenosť zaručuje, že nič relevantného sme neopomenuli, koherencia, že dohromady dávajú rozumný zmysel.

[7] Tu i ďalej budeme najčastejšie používať jazyk výrokovej logiky. Dôvod je ten, že sémantika stabilných modelov berie do úvahy miesto prvorádového programu množinu všetkých jeho základných inštancií, teda – vlastne – výrokovologický program, možno i nekonečný.

3.2 Reprezentácia poznatkov na úrovni údajových štruktúr a procedúr

V tejto časti uvedieme príklady, ktoré ukazujú, že procedurálny prístup, zameraný na manipulácie s údajovými štruktúrami, nie je adekvátny z hľadiska reprezentácia znalostí.

Algoritmus najkratšej cesty

V umelej inteligencii sa ako prostriedok na reprezentáciu znalostí intenzívne uplatňujú sémantické siete, v ktorých sa dedia znalosti z viacerých nadtried, môžu sa vyskytovať konflikty medzi nimi a pripúšťajú sa výnimky zo vzťahu podtrieda/nadtrieda. V takejto sieti možno odvodiť (reprezentovať!) vzájomne kontradiktorické tvrdenia. Folklórny je príklad o tvorovi menom Tweety, ktorý by mal lietať, pretože je vták a súčasne by nemal lietať, lebo je pštros.

Na riešenie týchto konfliktov vymyslel Scott Fahlman v sedemdesiatych rokoch minulého storočia algoritmus najkratšej cesty [7]. Tento algoritmus zisťoval, či (napríklad) Tweety lieta alebo nelieta tak, že odštartoval z vrcholu *Tweety* a paralelne prechádzal všetkými hranami krok po kroku k nasledujúcim vrcholom. Ak došiel k vrcholu *lieta* skôr po pozitívne označenej hrane, prijal sa záver, že Tweety lieta, ak prišiel skôr po negatívne označenej hrane, prijal sa záver, že Tweety nelieta. Ak naraz – zdržal sa úsudku (vo vtedajšej terminológii: obe cesty sa vzájomne neutralizovali).

Ako vidno, ide o čisto procedurálne definovaný prístup k inferencii. Tento algoritmus ako prostriedok na spracovanie poznatkov zlyháva. Náhodné hrany v alternatívnych reprezentáciách sieťami môžu podstatne ovplyvniť dĺžku cesty. V komplikovanejších sieťach niektoré vrcholy (na najkratšej ceste k cieľovému vrcholu) môžu byť neutralizované, a teda by nemali poskytovať podporu pre ďalšie závery.

Podstatné, čo kritici tohto algoritmu časom pochopili, bolo však to, že vrcholy (presnejšie – propozície, ktoré zodpovedajú vrcholom a hranám medzi nimi), ležiace na najkratšej ceste k cieľovému vrcholu, nemusia nutne tvoriť množinu hypotéz, ktoré dohromady dávajú zmysel. V technickejších termínoch – netvoria extenziu defaultovej teórie, ktorá zodpovedá danej sémantickej sieti.[8]

Videli sme, že čisto procedurálny prístup k manipulácii so štruktúrou nízkej úrovne nepostačuje na adekvátne spracovanie poznatkov, na korektné odpovedanie na otázky adresované báze poznatkov, lebo nevenuje dostatočnú pozornosť sémantickej špecifikácii usudzovania.

Sémantické siete - procedurálna a deklaratívna reprezentácia

Ku grafovým reprezentáciám sémantických sietí možno priradiť logický jazyk, v ktorom sa dá formulami vyjadriť presne to, čo v grafovom jazyku. A naopak, ľubovoľnú sémantickú sieť vyjadrenú v logickom jazyku, vhodnom na reprezentáciu hierarchií konceptov a ich rolí, možno ekvivalentne vyjadriť v grafovom jazyku. To vôbec nie je prekvapujúce. Iba o málo zaujímavejšie je, že

[8] Tradičné formalizmy nemonotónneho usudzovania – defaultová logika, cirkumskripcia, autoepistemická logika – možno preložiť do rôznych jazykov logického programovania. Preto je pozornosť tohto textu sústredená na logické programovanie ako na predstaviteľa systematického prístupu k reprezentácii znalostí a k usudzovaniu.

indexovanie, ktoré je prirodzene implicitné v sémantických sieťach (i rámcoch), možno zaviesť aj na formulách, pozri napríklad [20]. Preto aj ľubovoľný výpočet na grafovej reprezentácii možno previesť na výpočet na zodpovedajúcej logickej reprezentácii (a aj naopak).

Preto veľmi prekvapuje, keď niektorí výskumníci hovoria (prípadne hovorili) o grafovej reprezentácii sémantických sietí (kombinovanej s procedurálnymi inferenčnými technikami) ako o výpočtovo efektívnejšej, hoci menej rigorózne definovanej (v porovnaní s logickou reprezentáciou a "deklaratívnymi inferenčnými technikami"). Pozri napríklad [2], s. 320 (zdôrazňujeme, že ide o prvé vydanie tejto učebnice, frekventovane používanej na celom svete, v druhom vydaní sa táto pasáž už neobjavila).

Treba pripomenúť, že systematickému štúdiu reprezentácií toho typu, čo predstavujú sémantické siete a rámce, sa venujú logiky deskripcie (description logic). Usudzovanie pomocou logík deskripcie možno realizovať aj pomocou prekladu do AnsPrologu [5].

TMS

Ďalší príklad prístupu, opísaného a vytvoreného na (nízkej) úrovni, blízkej implementácii, je Truth Maintenance System [6]. Autor, Jon Doyle, tento prístup opísal na úrovni uzlov, zoznamov a procedúr, ktoré s nimi pracujú. Kľúčová bola Truth Maintenance procedúra, ktorej cieľom bolo udržiavať množinu presvedčení systému (množinu uzlov, ktoré sú IN – v pôvodnej formulácii) v adekvátnom stave po každej modifikácii. Doyle považoval túto procedúru za efektívnu. Po abstraktnejšom opise TMS v logickom jazyku s presnou sémantikou ukázal Elkan [9], že TM procedúra je vlastne procedúrou výpočtu stabilného modelu, čo je NP-úplný problém.

Zhrňme

Všetky uvedené príklady mali ukázať, že reprezentácia znalostí na úrovni údajových štruktúr a inferencia na úrovni procedúr bez sémantickej špecifikácie majú kritické nedostatky. Nezaručujú totiž korektné spracovania poznatkov a ich adekvátne využitie pri riešení problémov a realizovaní požiadaviek

Navyše, údajové štruktúry v jazyku bez jasne definovanej sémantiky sa obvykle v priebehu implementácie upravujú tak, aby sa ľahšie zvládli prípady, s ktorými sa vopred nerátalo. Podobne sa upravujú procedúry, ktoré nad týmito štruktúrami pracujú. Stráca sa prehľad o vlastnostiach a fungovaní systému. Sotva možno pri takomto prístupe riešiť fundamentálne problémy typu čo je odvoditeľné, či je implementované odvodenie korektné (a v akom zmysle) atď. V takom prípade nie je jasné, či odpovede na otázky sú správne, či rovnakú odpoveď na tú istú otázku nad tou istou bázou znalostí zaručene dostaneme aj pri ďalšom výpočte a pod. Ad hoc modifikácie procedurálne definovanej inferencie zatemňujú a komplikujú návrh reprezentácie, výpočty nad ňou a – predovšetkým – jej sémantiku.

4 Answer set programming

V posledných rokoch sa oblasťou intenzívneho výskumu stalo programovanie pomocou odpoveďových množín (ďalej uprednostníme anglický termín answer set programming). Ide o variant logického programovania (v rôzne silných jazykoch), ktorého cieľom je riešiť problémy a odpovedať na otázky pomocou odpoveďovej množiny – dobre definovanej sémantickej charakterizácie programu. Medzi hlavné aplikačné oblasti answer set programming patrí reprezentácia znalostí. Zaznamenania hodné je, že po krátkom období výskumu v oblasti[9] vychádza monografia [5]. V tejto monografii sa zavádza jazyk pre answer set programming, AnsProlog (presnejšie, ide o jeho rôzne silné verzie). Treba podotknúť, že z výpočtového hľadiska ide o ťažké problémy – napríklad problém, či daný konečný logický program má stabilný model, je NP-úplný. Je prirodzené, že ťažké problémy, súvisiace s usudzovaním a reprezentáciou znalostí, vyžadujú takýto nástroj. Výskum v oblasti venuje veľkú pozornosť optimalizácii výpočtov a dosiahnuté výsledky sú obdivuhodné: riešenia pre programy s desiatkami tisíc pravidiel sa dajú dosiahnuť v rozumnom čase (medzi 10 a 20 sekúnd).

Hlavným cieľom tejto časti je ukázať usudzovanie ako súčasť reprezentácie poznatkov a demonštrovať užitočnosť sémantiky odpoveďových množín (answer sets).

4.1 Jazyk normálnych logických programov

V jazyku normálnych logických programov sa používajú pravidlá, s ktorými sme sa už zoznámili – ide o pravidlá tvaru

$$A \leftarrow A_1, \ldots, A_k, \mathit{not}\ B_1, \ldots, \mathit{not}\ B_m,$$

kde A, A_i, B_j sú atómy.

Jedným zo základných problémov pre reprezentáciu znalostí sú vety charakterizujúce obvyklý stav vecí, typické prípady. Ide o druh znalostí, ktoré ľudia používajú intenzívne, ale ich reprezentácia pre výpočtové systémy nie je jednoduchá. V reprezentáciách pomocou sémantických sietí alebo rámcov sa takéto znalosti často vyjadrujú defaultami. Kritické je však to, že používanie defaultov vyžaduje istú globálnu kontrolu – neplatí, že ľubovoľné defaulty sú zlučiteľné. Ďalej, je výhodné, keď reprezentácia defaultov je prepojiteľná s reprezentáciou výnimiek. Navyše, je potrebné, aby reprezentácia výnimiek bola flexibilná: je vhodné, keď sa definícia výnimkových stavov môže dynamicky rozširovať (meniť).

[9] Stabilné modely (odpoveďové množiny) boli prvýkrát definované pred 16 (14) rokmi, [11, 12]. Prvé implementácie pochádzajú z polovice deväťdesiatych rokov [?,?], Pojem answer set programming propagujú [14, ?].

Už v jazyku normálnych logických programov možno vytvoriť reprezentáciu s uvedenými vlastnosťami. Obecná schéma je nasledovná:

$$
\begin{aligned}
type(X) &\leftarrow property(X), not\ exception(X) \\
exception(X) &\leftarrow case_1(X) \\
&\dots \\
exception(X) &\leftarrow case_k(X)
\end{aligned}
$$

Definujeme predikát *type*. Definícii vyhovuje každý objekt, ktorý má vlastnosť *property* a nie je výnimočným prípadom. Výnimočné prípady sa definujú zoznamom pravidiel. Samozrejme, tento zoznam možno podľa potreby meniť. Netreba azda pripomínať, že predpokladáme definície predikátov *property* a $case_i(X)$ pre každé i. Ak má takýto program aspoň jeden stabilný model, potom každý z jeho stabilných modelov predstavuje jednu koherentnú interpretáciu použitých predikátových symbolov. Ak máme viac stabilných modelov, môže to napríklad znamenať, že niektoré výnimočné stavy nemôžu nastať súčasne (povedzme, materská dovolenka a neukončená vojenská služba) alebo – opať napríklad – objekt o_1 môže mať vlastnosť *property* práve vtedy, keď ju nemá objekt o_2.

Stabilné modely teda poskytujú presnú sémantickú špecifikáciu defaultov a výnimočných stavov (vo vzťahu k nim). Výpočty nad takouto reprezentáciou (odpovedanie na dotazy) sú korektné vtedy, keď zodpovedajú takejto špecifikácii. Existenciu viacerých stabilných modelov netreba považovať za nedostatok: zodpovedajú alternatívnej povahe hypotetického usudzovania.[10]

Podobnou technikou sa reprezentuje známy problém rámca: Problémom je úsporne a adekvátne reprezentovať vplyv akcií na stav prostredia. Obvykle veci ostávajú po akcii v nezmenenom stave až na nejaké výnimky, ktoré treba explicitne špecifikovať. (Posunutie predmetu obvykle nemení jeho farbu, hmotnosť a pod. Treba však špecifikobvať, ako v danom prípade zmenilo jeho polohu.) Pokiaľ nejaká zmena nebola špecifikovaná, predpokladá sa, že platí to, čo platilo pred akciou.

Treba zdôrazniť, že v jazyku normálnych logických programov sa nejaký negatívny literál *not* A akceptuje vtedy, keď sa neakceptuje (nie je známe, že by platil) literál A. (Preto túto negáciu nazývame defaultovou.) Napríklad *not* $property(o)$ sa akceptuje vtedy, keď sa neakceptuje $property(o)$. Teda, pri tejto reprezentácii sa predpokladá univerzálna platnosť predpokladu uzavretého sveta. Tento nedostatok flexibility možno zmierniť v jazyku rozšírených logických programov.

[10] Sémantika stabilných modelov dáva podnet pre definovanie alternatívneho operátora odvodenia. Jeden z viacerých stabilných modelov nám môže predstavovať jedno z alternatívnych riešení. Tzv. dôverčivý prístup k definícii operátora odvodenia vyzerá tak, že nedeterministicky vyberáme niektorý zo stabilných modelov a preverujeme, akú odpoveď na položenú otázku poskytne.

4.2 Jazyk rozšírených logických programov

V jazyku rozšírených logických programov sa zavádza ďalší druh negácie, *explicitná* negácia $\neg$. Kým *not* A znamená, že nie je známe, že by platilo A, $\neg A$ má znamenať, že A neplatí.

Výskyt explicitnej negácie umožňuje jemnejšie narábanie s predpokladom uzavretého sveta. Tento predpoklad prijímame iba pre špecifikovaný predikátový symbol pomocou pravidla tvaru $\neg p(X) \leftarrow not\ p(X)$. To znamená, negatívny literál $\neg p(X)$ prijímame vtedy, keď nie je známe $p(X)$ – ak sme pre daný predikát špecifikovali explicitne túto podmienku. Pokiaľ ju nezavedieme explicitne pre predikátový symbol q, potom pre tento symbol predpoklad uzavretého sveta neplatí a $\neg q(X)$ akceptujeme iba vtedy, keď ho dokážeme odvodiť z daného programu (bez pomoci pravidla uzavretého sveta).

Jemnejšie narábanie s predpokladom uzavretého sveta umožňuje jemnejšiu prácu s typickými prípadmi a výnimkami.

Zovšeobecnením stabilných modelov pre rozšírené logické programy sú odpoveďové množiny (answer set). Pre jazyk rozšírených logických programov modifikujeme pojem literálu: Formuly tvaru A a $\neg A$, kde A je atóm, nazývame objektívnymi literálmi. Ak L je objektívny literál, *not* L je subjektívny literál. Literál je objektívny alebo subjektívny literál. Množina literálov je konzistentná, ak neobsahuje dvojicu A, $\neg A$ alebo dvojicu L, *not* L pre nejaký atóm A alebo objektívny literál L.

Konzistentnú množinu literálov S voláme *odpoveďovou množinou* výrokovo-logického programu P práve vtedy, keď je splnená nasledujúca podmienka: v Kripkeho štruktúre asociovanej s P existuje cesta, ktorej koreňom je S^-, množina všetkých subjektívnych literálov z S a ktorá je ukončená v S, t.j. neexistuje hrana $(S, w_\perp)$. (Kripkeho štruktúru asociovanú s rozšíreným programom definujeme analogicky ako pre normálne programy.)

Je zrejmé, že platí: ak S je odpoveďovou množinou rozšíreného programu P, potom pre ľubovoľný atóm A platí: do S patrí práve jedna z nasledujúcich troch dvojíc:

$$A, not\ \neg A \qquad \neg A, not\ A \qquad not\ A, not\ \neg A$$

Tretia dvojica dovoľuje explicitne narábať s pravdivostnou hodnotou *unknown*, prípadne s metapredikátom *unknown*.

4.3 Ďalšie rozšírenia

Sémantické siete a dedenie môžeme elegantne reprezentovať, ak do jazyka explicitne zavedieme popri pravidlách aj defaulty, ak vieme zaznamenať konflikty medzi defaultami a ak vieme vyjadriť, ktorý z defaultov preferujeme.

Tieto záležitosti reprezentujeme v jazyku, ktorý používa, iba fakty (pravidlá, ktorých pravá strana je prázdna). Ide o fakty nasledujúceho tvaru: $rule(r, l_0, [l_1, \ldots, l_m])$ – definujeme pravidlo r, jeho hlavou je literál l_0, telom množina literálov $l_1, \ldots, l_m$. Podobne pre defaulty: $default(d, l_0, [l_1, \ldots, l_m])$. Medzi dvojicami defaultov môžu byť konflikty: *conflict*(d_1, d_2) a môže byť definované, ktorý z konfliktných defaultov je preferovaný: *prefer*(d_1, d_2). Aj pre tento jazyk sa definuje sémantika odpoveďových množín.

Popri formalizácii sémantických sietí možno načrtnutý jazyk (s prioritami a riešením konfliktov) použiť aj na formalizáciu právneho usudzovania. V súvislosti so sémantickými sieťami pripomeňme, že už sme spomínali, že logiky deskripcie možno vyjadriť v AnsPrologu.

Dôležitou aplikáciou answer set programovania je usudzovanie o akciách a plánovanie. Definujú sa viaceré jazyky logického programovania špecializované na tento druh usudzovania, pozri [5] a robia sa početné výpočtové experimenty v tejto oblasti.

Ďalšie používané rozšírenia sú pravidlá s hlavou v tvare disjunkcie, s literálmi obsahujúcimi operátory poznania a racionálnej viery (knowledge, belief), pravidlá, pripúšťajúce aj defaultové negácie (atómov alebo aj všetkých objektívnych literálov) v hlave a dokonca aj pravidlá, obsahujúce v hlave disjunkcie ľubovoľných provorádových formúl a v tele konjunkcie ľubovoľných prvorádových formúl. Samozrejme, so zvyšujúcou sa vyjadrovacou silou rastie výpočtová zložitosť zodpovedajúceho formalizmu.

Pochopiteľne, jazyky logického programovania dovoľujú získať exaktné výsledky o výpočtovej zložitosti a vyjadrovacej sile týchto formalizmov, pozri [5].

Spomedzi ďalších aplikácií answer set programovania treba spomenúť aplikácie na abduktívne usudzovanie, na diagnostikovanie, debugovanie, na riešenie nekonzistentností (vrátane parakonzistentného usudzovania), na kontrafaktuálne usudzovanie, pozri [1, 5].

Najznámejšie implementácie systémov sú dlv [8], a smodels [17]. Výskum, na ktorom sú tieto implementácie založené, viedol k sofistikovaným algoritmom, využívajúcim silné heuristiky a optimalizácie. Výskum ešte zďaleka nemožno považovať za uzavretý. V súčasnosti sa doňho zapája v nadväznosti na výskum Skupiny znalostných systémov na Technickej univerzite vo Viedni aj naše pracovisko v rámci grantu, získaného v Akcii Rakúsko-Slovensko.

4.4 Logika dynamických báz znalostí

Ako pozoruhodnú aplikáciu sémantiky stabilných modelov uvedieme logické výskumy v oblasti dynamického logického programovania. Dynamika báz poznatkov nie je ľahko prístupná logickému výskumu. Ak odhliadneme od izolovaných pokusov (napríklad Suszko v šesťdesiatych rokoch), prvým signálom zmeny záujmov sa stal logický výskum dynamiky poznania v štýle postulátov racionality, ktorý iniciovali Alchourrón, Gärdenfors a Makinson, pozri [10].

Podstatne priliehavejším modelom báz poznatkov sú však multidimenzionálne dynamické logické programy. Ide o množinu logických programov v jazyku, ktorý pripúšťa oba druhy negácií a v hlave pravidiel môžu byť aj defaultové negácie objektívnych literálov. Na týchto programoch je definovaná acyklická relácia preferencie.

Tu si budeme všímať iba postupnosti programov (neskoršie programy v postupnosti sú viac preferované – predstavujú aktuálnejšiu informáciu). Medzi programami môžu vznikať konflikty. Ide o konflikty medzi hlavami pravidiel (jedna je napríklad A a druhá $\neg A$ alebo $not\ A$). Tie sa riešia tak, že sa uprednostní pravidlo neskoršieho programu a odmietne pravidlo skoršieho programu. Pre prípad dvoch programov – P , pôvodného programu a U, modifikujúceho programu – sa sémantika programu získaného po modifikácii definuje ako dynamický stabilný model. Je to množina literálov S, spĺňajúca túto podmienku: $S = least(U \cup (P \setminus reject(P, S)) \cup defaults(P \cup U, S)$, kde *least* je operátor, ktorý produkuje najmenší model (program sa chápe ako definitný – všetky negácie sa berú ako nové atómy), $reject(P, S)$ je množina pravidiel, programu P, ktoré treba odmietnuť vzhľadom na S a $defaults(P \cup U, S)$ je nožina defaultových predpokladov, ktoré možno prijať pri S. Teda, najmenší model sa ráta pre pravidlá z U, doplnené o neodmietnuté pravidlá z P a o defaultové predpoklady. Detaily týkajúce sa defaultov a odmienutých pravidiel obídeme, možno ich nájsť napríklad v [13].

Sémantická problematika evolvujúcich báz poznatkov však naráža na mnohé vážne problémy. Tvrdým orieškom sú cyklické modifikácie (modifikujúci program obsahuje cyklické závislosti medzi literálmi).Niektoré typy konfliktov však principiálne nie sú riešiteľné, ak sa pozornosť venuje iba konfliktom medzi pravidlami.

Ak napríklad program P je $a \leftarrow b, c$ a program U je

$$b \leftarrow not\ a$$
$$c \leftarrow b,$$

nenastáva nijaký konflikt medzi pravidlami, ale je tu očividný konlikt medzi závislosťou literálov z množiny $w = \{b, c\}$ od literálu $not\ a$, generovanou preferovaným programom a závislosťou atómu a na w, generovanou menej preferovaným programom. Nepríjemnej (a neakceptovateľnej) závislosti a na $not\ a$ sa môžeme zbaviť tak, že miesto pravidiel odmietame hrany v Kripkeho štruktúrach, ktoré sme definovali v časti 4. Ukazuje sa, že táto sémantika dynamických logických programov je nielen schopná jemnejšieho rozlíšenia, ale vysporiadava sa aj s problémom cyklických modifikácií, pozri [22].

5 Záver

Logické formalizmy majú vlastnosť, ktorá je mimoriadne dôležitá z hľadiska reprezentácie znalostí. Ide o to, že dovoľujú relatívne pohodlné adaptácie, modifikácie jazyka na reprezentáciu znalostí s cieľom zosilniť ich tak, aby zvládali ďalšie úlohy. Mnohé výrazné vlastnosti spred modifikácie ostávajú nedotknuté. Táto schopnosť sa nazýva v angličtine elaboration tolerance.

V texte sme stručne charakterizovali hlavné zisky z investícií, venovaných starostlivej sémantickej špecifikácii jazykov na reprezentáciu znalostí a ich usudzovacích schopností.

Poďakovanie. Autor ďakuje jednému z anonymných recenzentov za podrobné a cenné pripomienky.

Literatúra

1. Alferes, J., Pereira, L. *Reasoning with logic programming.* Springer 1996
2. Allen, J. *Natural Language Understanding.* Benjamin/Cummings Publ.Comp.1987
3. Artificial Intelligence, vol. 13, No 1-2
4. Baral, C., Gelfond, M. *Logic Programming and Knowledge Representation.* Journal of Logic Programming, 1994
5. Baral, C. *Knowledge representation, reasoning and declarative problem solving.* Cambridge 2003
6. Doyle, J. *A truth maintenance system.* Artificial Intelligence 12:231-272, 1979
7. Fahlman, S. *NETL: A System for Representing and Using Real World Knowledge.* MIT Press, Cambridge, MA, 1979
8. Eiter, T. et al. *The dlv system.* In Minker (ed.), Pre-prints of workshop on Logic-based AI. 2000
9. Elkan, C. *A rational reconstruction of non-monotonic truth maintenance systems.* Artificial Intelligence, 43(2):219-234, 1990
10. Gärdenfors, P., Rott, H. *Belief Revision.* In: Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 4, Epistemic and Temporal Reasoning, Oxford 1995
11. Gelfond, M., Lifschitz, V. *The stable model semantics for logic programming.* Proc. ICLP 1988, 1070-1080
12. Gelfond, M., Lifschitz, V. *Logic programs with classical negation.* Proc. ICLP 1990, 579-597
13. Leite, J. *Evolving knowledge bases.* IOS Press 2003.
14. Marek, W., Truszczynski, M. *Stable models and an alternative logic programming paradigm.* In The logic programming paradigm a 25-year perspective, 375-398, Springer, 1999
15. Mc Carthy, J. *Programs with common sense,* 1959, `http://www-formal.stanford.edu/jmc/mcc59.html`
16. Minski, M. *A Framework for Representing Knowledge.* MIT, (1974)
17. Niemelä, I., Simons, P. *Efficient implementation of the well-founded and stable model semantics.* Proc. ICLP 1996, 289-303
18. Niemelä, I. *Logic programs with stable models semantics as a constraint programming paradigm.* Annals of Matahematics and Artificial Intelligence, 1999, 241-271
19. Przymusinski, T. *Non-monotonic formalisms and logic programming.* Proc. of 6th Int. Conf. on Logic Programming. MIT Press 1989, 655-674
20. Šefránek, J. *Inteligencia ako výpočet.* IRIS Bratislava 2000
21. Šefránek, J. *A Kripkean semantics for dynamic logic programming.* LPAR 2000, Springer, 469-486
22. Šefránek, J. *Rejection of dependencies vs. rejection of rules.* http://www.ii.fmph.uniba.sk/ sefranek/online/rejects.ps