Negativní informace

Petr Štěpánek

S použitím materiálu M.Gelfonda a V. Lifschitze

2009

Negace jako neúspěch

Motivace: Tvrzení *p* (atomická formule) neplatí, jestliže nelze odvodit žádnou jeho instanci *p'*.

V Prologu negaci definovanou neúspěchem vyjadřuje operátor `not`, který je interně definován dvěma klauzulemi

```
not(p) :-  p,!, fail.
not(p).
```

Ty dávají výrazu `not(p)` následující operační význam

Jestliže cíl `p` uspěje, na dotaz `not(p)` dává odpověď `no`,

jestliže `p` neuspěje, na dotaz `not(p)` odpoví `yes`.

Tedy `not(p)` nevrací žádnou hodnotu jenom potvrzení `yes` nebo `no.`

Proto `not(p)` chápeme jenom jako " testovací" cíl. V tom se liší od logické negace `¬p .` Ukazuje to následující

Příklad. *Situaci v hotelu popisují dva typy tvrzení:*

Hotel je obsazen

Pokoj(13) je volný

Pokoj(313) je volný

V Prologu například

```
hotel_obsazen :- not(volný_pokoj(X)).
volny_pokoj(13).
volny_pokoj(313).
```

Můžeme očekávat konverzaci

```
?- hotel_obsazen.
no
?- not(hotel_obsazen).
yes
?- volny_pokoj(X).
X = 13;
X – 313;
no
```

Uvedený program však nepřipouští dotaz "*které pokoje nejsou volné ?*"

V Prologu mohou být negované cíle použity jen jako testovací. Negace jako neúspěch tedy není logická negace. Platí:

• je-li atom `p` základní a `not(p)` uspěje znamená to, že cíl `p` nelze odvodit, volně řečeno, že atom `p` není pravdivý,

• má-li atom `p` proměnné, $X_1, \ldots, X_n$, a cíl `not(p)` uspěje, znamená to, že nelze odvodit žádnou instanci `p'`.

Definice. Je-li *P* program v Prologu,

(i) Říkáme, že prologovský strom *konečně selhává*, je-li konečný a neobsahuje prázdnou klusuli,

(ii) atom *A konečně selhává vzhledem k P* , je-li kořenem konečně selhávajícího stromu .

Tedy všechny větve konečně selhávajícího stromu jsou neúspěšné derivace programu P a atom A konečně selhává jestliže všechny derivace pro $P \cup \{A\}$ jsou neúspěšné (tedy také konečné).

Jinými slovy, cíl `not(A)` uspěje, právě když atom A konečně selhává.

Poznámka. Negace jako neúspěch jen ukazuje, že procedurální interpretace prologovských programů používá tzv. *Hypotézu uzavřeného světa.*

$$\frac{A \text{ nelze odvodit z } P}{\texttt{not(A)}}$$

Prologovské programy, které používají operátor negace jako neúspěch se nazývají obecné programy.

Definice. Říkáme, že P je *obecný program* jestliže sestává z definitních klauzulí a klauzulí tvaru

$$A \leftarrow B_1 , B_2 , \ldots , B_m , not(B_{m+1}), not(B_{m+2}), \ldots , not(B_n) \qquad (1)$$

kde $n \geq m \geq 0$ a A a každé B_1 je atom.

Obecný program rozděluje množinu základních dotazů na dvě množiny: na základní dotaz odpovídá `yes` nebo `no` .

Každý základní atom, který nevyplývá z faktů uvedených v programu se považuje za nepravdivý v souladu s Hypotézou uzavřeného světa.

Procedurální metoda vyhodnocení dotazů v obecných programech dává odpověď `no` na každý dotaz, který neuspěje.

Naproti tomu konzistentní teorie v predikátové logice prvního řádu dělí uzavřené formule (sentence) na tři skupiny:

Sentence je buď dokazatelná nebo nedokazatelná nebo nerozhodnutelná

Přitom nerozhodnutelnost se obvykle chápe jako neúplnost informací, které teorie poskytuje pro řešení problému.

Ve srovnání s predikátovou logikou, definitní ani obecné logické programy nedovolují pracovat přímo s neúplnou informací. Abychom překonali takové omezení, rozšíříme třídu obecných logických programů přidáme klasickou negaci $\neg$ k negaci definované neúspěchem `not` .

Tím zavedeme třídu rozšířených programů. Obecné logické programy poskytují negativní informaci implicitně pomocí pravidla uzavřeného světa, zatím co rozšířený program může vyjádřit negativní informaci explicitně.

V jazyce rozšířených programů můžeme rozlišovat mezi dotazy, které neuspějí, protože odvození odpovědi na dotaz je neúspěšná derivace a dotazy, které neuspějí, protože jejich negace uspěje.

Připomeňme, že literál je formule tvaru A nebo $\neg A$.

Definice. Říkáme, že P je *rozšířený program* jestliže sestává z klauzulí tvaru

$$L_0 \leftarrow L_1 , L_2 , \ldots , L_m , not(L_{m+1}), not(L_{m+2}), \ldots , not(L_n) \qquad (2)$$

kde $n \geq m \geq 0$ a každé L_i je literál.

Pro základní dotaz A rozšířený program dává odpověď `yes`, `no` nebo `unknown` podle toho jestli množina odpovědí obsahuje A, $\neg A$ nebo žádný z obou literálů. Odpověď `no` odpovídá explicitní negativní informaci obsažené v rozšířeném programu.

Klasická negace.

Uvažujme rozšířený program $\boldsymbol{P}_1$ sestávající z jediné klazule

$$\neg Q \leftarrow not\ P$$

Intuitivně ji chápeme jako tvrzení 'Q neplatí (je nepravdivé) jestliže nelze ukázat, že P je pravdivé'. Později ukážeme, že $\{\neg Q\}$ je množina odpovědí pro tento program. Odpovědi, které tento program dává pro dotazy P, Q jsou `unknown` a `no.`

Srovnejme dva rozšířené programy, které neobsahují `not.`

Program $\boldsymbol{P}_2$

$$\neg P \leftarrow .$$
$$P \leftarrow \neg Q$$

Program $\boldsymbol{P}_3$

$$\neg P \leftarrow .$$

$$Q \leftarrow \neg P$$

Potom $\{\neg P\}$ je množina odpovědí pro $\boldsymbol{P}_2$ a $\{\neg P, Q\}$ je množina odpovědí pro $\boldsymbol{P}_3$.

Naše sémantika není ‘kontrapozitivní’ vzhledem k $\leftarrow$ a $\neg$; dává různé odpovědi k programovým klauzulím $P \leftarrow \neg Q$ a $Q \leftarrow \neg P$, které jsou v predikátové logice ekvivalentní. To proto, že tato sémantika interpretuje obě klauzule jako odvozovací pravidla a ne jako implikace.

Jazyk rozšířených programů obsahuje klasickou negaci, ale neobsahuje klasickou implikaci.

Tento přístup je z výpočtového hlediska výhodný. Za dosti obecných předpokladů se dá ukázat, že vyhodnocení klauzule rozšířeného programu se dá převést na vyhodnocení dvou klauzulí, které neobsahují klasickou negaci. Tedy rozšíření obecných programů nepřináší žádné nové požadavky na výpočty.

Odpovědní množiny (Answer Sets)

Sémantika rozšířených programů chápe klauzuli s proměnnými jako zkratku za množinu pravidel, která odpovídají všem jejím základním instancím. Proto stačí definovat odpovědní množiny pro rozšířené programy bez proměnných. To uděláme ve dvou krocích.

Nejprve uvažujeme rozšířené programy bez proměnných, které navíc neobsahují *not* tj. $m = n$ v každé klauzuli (2). Takové jsou oba programy $\boldsymbol{P}_2$ a $\boldsymbol{P}_3$.

Rozšířené programy bez *not* volně řečeno odpovídají obvyklým logickým programům s tím, že místo atomů vystupují literály. Takové programy budou mít jen jednu odpovědní množinu. Prvky této množiny budou základní literály generované dvěma neformálními pravidly

(i) klauzulemi programu, a
(ii) klasickou logikou .

Z klasické logiky vyplývá, že z množiny základních literálů lze odvodit literál, který do této množiny nepatří, jen když tato množina obsahuje dva komplementární literály A a $\neg A$.

Uvedené pozorování pak motivuje následující definici odpovědní množiny.

Definice. (Odpovědní množina 1) Nechť $\boldsymbol{P}$ je rozšířený program, který neobsahuje `not` a nechť Lit je množina základních literálů v jazyce programu $\boldsymbol{P}$. Odpovědní množina pro $\boldsymbol{P}$ je minimální podmnožina S množiny Lit, která splňuje následující dvě podmínky

(i) Pro každé pravidlo $L_0 \leftarrow L_1, L_2, \ldots, L_m$ programu $\boldsymbol{P}$ takové, že literály $L_1, L_2, \ldots, L_m$ patří do S, potom L_0 je také prvkem S,

(ii) pokud S obsahuje komplementární pár literálů, pak $S = Lit$.

Odpovědní množinu programu $\boldsymbol{P}$, který neobsahuje negaci jako neúspěch budeme označovat $\alpha(P)$.

Pokud $\boldsymbol{P}$ neobsahuje `not` ani $\neg$, podmínka (ii) je triviální a $\alpha(\boldsymbol{P})$ je nejmenší Herbrandův model $\boldsymbol{P}$.

Je také zřejmé, že odpovědní množiny pro programy $\boldsymbol{P}_2$ a $\boldsymbol{P}_3$ uvedené v motivaci jsou odpovědní množiny podle definice Odpovědní množina 1.

Platí $\alpha(\boldsymbol{P}_2) = \{\neg P\}$ a $\alpha(\boldsymbol{P}_3) = \{\neg P, Q\}$.

Nyní mějme rozšířený program $\boldsymbol{P}$ bez proměnných. Označme množinu základních literálů v jazyce programu $\boldsymbol{P}$ symbolem *Lit*. Pro každou množinu $S \subseteq Lit$ označme $\boldsymbol{P}^S$ rozšířený program, který vznikne z $\boldsymbol{P}$ vynecháním

(i) každého pravidla, které obsahují v těle formuli `not`L , L je prvkem S
(ii) a všechny formule `not`L v tělech ostatních pravidel .

Definice. (Odpovědní množina 2) $\boldsymbol{P}^S$ již neobsahuje `not` a jeho odpovědní množina byla definována jako Odpovědní množina 1 . Pokud je tato odpovědní množina rovna S, řekneme, že S je odpovědní množina pro $\boldsymbol{P}$.

Jinými slovy, odpovědní množina pro $\boldsymbol{P}$ je charakterizována rovnicí

$$S = \alpha(\boldsymbol{P}^S) \qquad (3)$$

Abychom ověřili, že $\{\neg Q\}$ je odpovědní množina pro program $\boldsymbol{P}_1$ je třeba sestrojit program $\boldsymbol{P}_1^{\{\neg Q\}}$. Ten sestává z jediného pravidla

$$\neg Q \leftarrow .$$

Toto pravidlo vznikne vynecháním *not P* z jediného pravidla $\boldsymbol{P}_1$. Odpovědní množina $\boldsymbol{P}_1^{\{\neg Q\}}$, a tedy $\boldsymbol{P}_1$ je vskutku $\{\neg Q\}$. Je zřejmé, že žádná jiná množina literálů S nesplňuje (fixpointovou) podmínku (3).

Ještě je třeba ověřit, že druhá, obecnější definice odpovědních množin, pokud je použita k programu bez *not* je ekvivalentní s první definicí.

To bezprostředně plyne *z* faktu, že pro takový program $\boldsymbol{P}$ platí

$$\boldsymbol{P}^{S} = \boldsymbol{P}$$

takže z rovnosti (3) dostáváme

$$S = \alpha(\boldsymbol{P})$$

Na druhé straně, jestliže $\boldsymbol{P}$ neobsahuje $\neg$, pak je to obecný logický program a $\boldsymbol{P}^{S}$ je obyčejný logický program. Jeho odpovědní množina neobsahuje negativní literály.

Proto odpovědní množina obecného logického programu (neobsahuje klasickou negaci) je množina atomů.

Odpovědní množiny můžeme také považovat za neúplné teorie v logice ve smyslu tamní definice. Tedy pro odpovědní množinu S pro nějaký program $\boldsymbol{P}$ a nějakou formuli A v jazyce programu $\boldsymbol{P}$ obecně nemusí platit

$$S \mid\!\!- A \quad \text{nebo} \quad S \mid\!\!- \neg A$$

Má-li program několik odpovědních množin, je neúplný v jiném smyslu: má několik různých interpretací a odpověď na dotaz může záviset na interpretaci.

Rozšířený program je sporný pokud má spornou odpovědní množinu, tedy odpovědní množinu obsahující komplementární pár literálů.

Takový je například program $\boldsymbol{P}_4$ sestávající z pravidel

$$P \leftarrow .$$

$$\neg P \leftarrow .$$

Je zřejmé, že obecné logické programy nemohou být sporné.

Pozorování. Sporný rozšířený logický program má právě jednu odpovědní množinu - množinu všech literálů *Lit* .

Důkaz. Z definice odpovědních množin je zřejmé, že každá odpovědní množina, která obsahuje komplementární pár literálů se rovná *Lit*. Fakt že sporný program nemá jinou odpovědní množinu plyne z obecnějšího tvrzení.

Pozorování. Rozšířený program nemůže mít dvě odpovědní množiny takové, že jedna z nich je vlastní podmnožinou druhé.

Důkaz. Předpokládejme, že S a S' jsou dvě odpovědní množiny programu $\boldsymbol{P}$ takové, že $S \subset S'$. Potom $\boldsymbol{P}^{S'} \subseteq \boldsymbol{P}^{S}$ odkud plyne $\alpha(\boldsymbol{P}^{S'}) \subseteq \alpha(\boldsymbol{P}^{S})$, tedy $S' \subseteq S$ a $S' = S$.

Bezespornost sama nezaručuje existenci odpovědní množiny. Obecný logický program $\boldsymbol{P}_5$

$$P \leftarrow not\ P$$

Je toho dokladem.

Odbočka. (Multiagentní systémy) Intuice napovídá, že odpovědní množiny mohou být vhodnými množinami přesvědčení, které racionální agent může mít na základě informací vyjádřených pravidly programu $\boldsymbol{P}$.

Je-li S množina základních literálů o kterých je agent převědčen, že jsou pravdivé, potom žádné pravidlo, které obsahuje cíl $not\ L$ pro literál L z množiny S pro něj není užitečné a každý cíl $not\ L$, kde L patří do S bude pro něj trivální.

Bude tedy schopen nahradit množinu pravidel programu $\boldsymbol{P}$ množinou zjednodušených pravidel programu $\boldsymbol{P}^S$. Je-li odpovědní množina pro $\boldsymbol{P}^S$ totožná s S , potom volba S jako množiny přesvědčení je racionální.