Representace znalostí
s použitím klasické negace

Petr Štěpánek

S využitím materiálu M. Gelfonda a V. Lifschitze

2009

Negace jako neúspěch v logických programech vede v některých případech k nežádoucím výsledkům.

Příklad. (Autobus na přejezdu, John McCarthy) Autobus smí přejet přejezd pokud se neblíží žádný vlak . To se dá vyjádřit pravidlem (klauzulí):

$$\textit{Přejeď} \quad \leftarrow \quad \textit{not Vlak.}$$

jestliže je atom *vlak* interpretován jako nepřítomnost blížícího se vlaku.

Ovšem taková reprezentace znalostí je nepřijatelná, pokud připustíme, že informace o přítomnosti nebo nepřítomnosti vlaku nemusí být dostupná. Jestliže atom *Vlak* není v databázi například proto, že strojvedoucí má zastíněný výhled, pak jistě nechceme, aby autobus přejížděl koleje.

Situace se změní, použijeme-li klasickou negaci:

$$Přejeď \leftarrow \neg Vlak.$$

Potom literál *Přejeď* nebude prvkem odpovědní množiny dokud negativní fakt $\neg$ *Vlak* není v databázi.

Rozdíl mezi *not P* a $\neg P$ v logickém programu je podstatný pokud nemůžeme předpokládat, že dostupná pozitivní informace o *P* je úplná, tj. když hypotéza uzavřeného světa není použitelná k predikátu *P*.

Hypotéza uzavřeného světa pro predikát *P* se dá vyjádřit v jazyce rozšířených programů klauzulí

$$\neg P(x) \leftarrow not\ P(x) \qquad (1)$$

Pokud je klauzule (1) v programu, pak *not P*(x) a $\neg$ *P*(x) jsou zaměnitelné. Jinak používáme *not P* k vyjádření toho, že není známo zda *P* je pravdivé , a $\neg P$ k vyjádření, že *P* je nepravdivé.

Pro některé predikáty může být výhodný předpoklad vyjádřený ‘kontrapositivní‘ klauzulí

$$P(x) \leftarrow not \neg P(x) \qquad (2)$$

Pak bude možné definovat množinu koncových vrcholů orientovaného grafu programem $\boldsymbol{P}_1$:

$$\neg Terminal(x) \leftarrow not\ Arc(x,y).$$
$$Terminal(x) \leftarrow not \neg Terminal(x).$$

Pro daný predikát P si můžeme vybrat zda do programu vložíme klauzuli (1) nebo (2) nebo žádnou z nich.

Příklad. (Stanford a SRI) Jack je zaměstnán na Stanfordské Univerzitě a Jane na SRI International:

$$Employed(Jack,Stanford) \leftarrow .$$
$$Employed(Jane,SRI) \leftarrow .$$

Zaměstnané osoby mají adekvátní plat:

$$Adequate\text{-}Income(x) \leftarrow Employed(x,y) \ . \qquad (3)$$

Odpovědní množina pro tento program:

$$\{Employed(Jack,Stanford), Employed(Jane,SRI),$$

$$Adequate\text{-}Income(Jack),\ Adequite\text{-}Income(Jane)\}.$$

Tato množina neobsahuje negativní literály, ale nedovoluje odvodit, že Jack není zaměstnán na SRI.

Požadavek, aby informace o zaměstní byla v databázi úplná, můžeme splnit tím, že vyjádříme předpoklad uzavřeného světa pro predikát *Employed*:

$$\neg Employed(x,y) \leftarrow not\ Employed(x,y).$$

Přidáme-li toto pravidlo k programu, přidáme k odpovědní množině literály

$$\neg Employed(Jack,SRI)\ ,\ \neg Employed(Jane,Stanford)$$

Je-li dostupná informace o zaměstnancích úplná jen pro, ale ne pro SRI, přidáme jen následující instanci klauzule (3):

$$\neg Employed(x,Stanford) \leftarrow not\ Employed(x,Stanford). \qquad (4)$$

Následující příklad použití negace v Zákonu o britském občanství (1981 Britisht Nationality Act) cituje Kowalski: „Ode dne platnosti toho zákona žádná osoba nebude mít status občana Commonwealthu ani status Britského poddaného jinak než podle tohoto zákona.“

Toto ustanovení v podstatě postuluje Hypotézu uzavřeného světa pro některé predikáty. Kowalski poznamenává, že není třeba taková ustanovení uvádět v obecných logických programech, protože jsou implicitně obsaženy v sémantice jazyka. Zdá se však, že souhlasí s tím, že může být užitečné připustit predikáty, definice kterých se nepovažují za úplné, a požadovat „explicitní deklaraci“ kdykoliv se činí předpoklad Hypotézy uzavřeného světa. V jazyce rozšířených programů se takové deklarace representují pravidly tvaru (1).

Ještě jeden příklad ve kterém se vyskytují oba typy negace. V zadání se objevují anglické termíny: *GPA* je celkový psychologiký test, *Interview* je v tomto případě pohovor před stipendijní komisí, *Minority* je status minoritní skupiny obyvatel a *Eligible* znamená vhodný, přicházející v úvahu.

Příklad (College X). Pro udělování stipendii studentům na uvedené škole se používají následující pravidla:

1. Každý student s GPA nejméně 3,8 přichází v úvahu.
2. Každý minoritní student s GPA nejméně 3,6 přichází v úvahu.
3. Žádný student s GPA menším než 3,6 nepřichází v úvahu pokud úspěšně neprojde pohovorem před stipendijní komisí.
4. Studenti, jejichž vhodnost není určena pravidly 1.-3. projdou pohovorem.

Uvedená pravidla jsou zakódována v následujícím rozšířeném programu:

$$
\begin{aligned}
&Eligible(x) \leftarrow HighGPA(x).\\
&Eligible(x) \leftarrow Minority(x), FairGPA(x).\\
&\neg Eligible(x) \leftarrow \neg FairGPA(x).\\
&Interview(x) \leftarrow \text{not } Eligible(x), \text{not } \neg Eligible(x).
\end{aligned}
$$

Předpokládejme, že následující fakta jsou k dispozici o jedné studentce:

$$FairGPA(Ann) \leftarrow . \neg HighGPA(Ann).$$

Rozšířený program $\boldsymbol{P}_1$ sestávající z těchto šesti pravidel má jedinou odpovědní množinu :

$$\{FairGPA(Ann)\ ,\ \neg\ HighGPA(Ann),\ Interview(Ann)\}\ . \qquad (5)$$

Databáze neobsahuje informaci, že *Ann* je minoritní studentka, která z principu takový údaj z žádosti neuvádí.

Interpretace obecných programů pomocí uzavřeného světa.

Syntakticky jsou obecné logické programy speciálním případem rozšířených programů. O všem je rozdíl jestliže obecný program interpretujeme jako obecný nebo jako rozšířený. Ne-li základní atom A prvkem odpovědní množiny obecného programu, znamená to, že korektní odpověď na dotaz A je *no* . Nepřítomnost základního atomu A v odpovědní množině rozšířeného programu, který sestává ze stejných klauzulí, znamená, že odpověď na dotaz A musí být *unknown*.

Příklad (Sudé numerály). Odpovědní množina programu

$$Even(0) \leftarrow .$$
$$Even(S(S(x)) \leftarrow Even(x).$$

je

$$\{Even(0), Even(S(S(0))), \ \ldots\}.$$

Protože tato množina neobsahuje $\neg\, Even(S(0))$ ani $Even(S(0))$, sémantika rozšířených programů říká, že odpověď na dotaz *Even(S(0))* je *unknown* - jinak než byl zamýšlený význam uvedené definice predikátu *Even* . Tento význam je možné vyjádřit v jazyce rozšířených programů tím, že přidáme předpoklad uzavřeného světa pro *Even :*

$$\neg\, Even(x) \leftarrow not\ Even(x).$$

Tento příklad ukazuje, že „sémanticky ekvivalentní“ rozšířený program k programu ***P*** lze získat tím, že k ***P*** přidáme předpoklad uzavřeného světa pro každý z predikátů.

Definice. (**R**ozšíření CW(***P***) obecného programu ***P***) Interpretace CW(***P***) obecného programu ***P*** jako rozšířeného programu s explicitním vyjádřením hypotézy uzavřeného světa vznikne z ***P*** přidáním pravidel

$$P(x_1,x_2\,,\ \dots\ x_n) \leftarrow not\ \neg\, P(x_1,x_2\,,\ \dots\ x_n)$$

Pro všechny predikátové symboly P , kde proměnné x_i jsou navzájem různé a P je n –ární.

Následující věta ukazuje, že odpovědní množiny programu CW($\boldsymbol{P}$) jsou v takovém vztahu k odpovědním množinám programu $\boldsymbol{P}$ jaký požadujeme. Proto označme Pos množinu všech positivních základních literálů v jazyce programu $\boldsymbol{P}$.

Věta. Je-li S odpovědní množina obecného logického programu $\boldsymbol{P}$, potom

$$S \cup \{\neg A \mid A \in (Pos - S)\} \tag{6}$$

je odpovědní množina CW($\boldsymbol{P}$). Navíc každá odpovědní množina programu CW($\boldsymbol{P}$) může být vyjádřena ve tvaru (6), kde S je odpovědní množina $\boldsymbol{P}$.

Důkaz. Bez újmy na obecnosti můžeme předpokládat, že $\boldsymbol{P}$ neobsahuje proměnné. Nechť program $\boldsymbol{P}'$ vznikne z $\boldsymbol{P}$ tím, že nahradíme všechna pravidla v CW($\boldsymbol{P}$) jejich základními instancemi. Potom

$$\boldsymbol{P}' = \boldsymbol{P} \cup \{\neg A \leftarrow not\ A \mid A \in Pos\}$$

První tvrzení Věty

Je-li S odpovědní množina programu $\boldsymbol{P}$ je třeba dokázat, že množina (6) je odpovědní množinou $\boldsymbol{P}'$. Označme množinu (6) symbolem S'. Podle definice programu $\boldsymbol{P}^S$ dostáváme

$$(\boldsymbol{P}')^{S'} = \boldsymbol{P}^S \cup \{\neg A \leftarrow \mid A \in (Pos - S)\}$$

Odtud plyne

$$\alpha((\boldsymbol{P}')S') = \alpha(\boldsymbol{P}^S) \cup \{\neg A \mid A \in (Pos - S)\}$$

$$= S \cup \{\neg A \mid A \in (Pos - S)\} = S'$$

takže S' je odpovědní množinou pro $\boldsymbol{P}'$. Tím je dokázáno první tvrzení Věty.

Druhé tvrzeni Věty

Abychom dokázali druhé tvrzení, mějme libovolnou odpovědní množinu S' pro $\boldsymbol{P}'$ a definujme $S = S' \cap Pos$.
Potom platí

$$S' = \alpha((\boldsymbol{P}')^{S'}) = \alpha((\boldsymbol{P})^{S}) \cup \{\neg A \mid A \in (Pos - S')\}$$
$$= \alpha((\boldsymbol{P})^{S}) \cup \{\neg A \mid A \in (Pos - S)\}$$

První člen sjednocení je positivní část S' a druhý člen sjednocení je negativní část S'. Odtud nejprve dostaneme

$$\alpha((\boldsymbol{P})S) = S' \cap Pos = S$$

tj. S' je odpovědní množina pro $\boldsymbol{P}$ a dále

$$\{\neg A \mid A \in (Pos - S)\} = S' - Pos$$

Nakonec dostaneme

$$S \cup \{\neg A \mid A \in (Pos - S)\} = S \cup (S' - Pos)$$

$$= (S' \cap Pos) \cup (S' - Pos)$$

$$= S'$$

Pro libovolnou odpovědní množinu pro $\boldsymbol{P}' = \mathrm{CW}(\boldsymbol{P})$ jsme ukázali, že ji lze vyjádřit ve tvaru (6). Tím je druhé tvrzení a celá Věta dokázána.

Stratifikované programy. Tento pojem bude dále několikrát zmiňován i když jen v komentářích. Zde je definice:

Je-li ***P*** rozšířený program a st(•): $Pred_P : \rightarrow N$ je zobrazení množiny predikátů v ***P*** do přirozených čísel takové, že pro každé pravidlo a predikát *p* v jeho hlavě a *q* v těle platí

$$\begin{array}{ll} st(p) \geq st(q) & \text{je-li } q \text{ v pozitivním literálu} \\ st(p) > st(q) & \text{je-li } q \text{ v negativním literálu} \end{array}$$

Zde za negativní literál považujeme i výraz *not L* , který ani literálem není.

Potom říkáme, že program ***P*** je stratifikovaný. Zobrazení st(•) je stratifikací programu ***P*** . Pokud takové zobrazení neexistuje, říkáme, že program ***P*** není stratifikovaný.

Jinými slovy, stratifikace je takové očíslování predikátů, které vylučuje, aby se rekurze používala na kterýkoli typ negace.

Klasická negace v Disjunktivních databázích

Disjunktivní databáze a disjunktivní logické programování tvoří samostatnou kapitolu. Nebudeme se jí podrobně zabývat, omezíme se jen na jednoduchý příklad.

Příklad. (Stanford a SRI ještě jednou). Uvažujme tvrzení: Jack je zaměstán na Stanfordově Univerzitě nebo v SRI ; každá zaměstnaná osoba má adekvátní příjem. Intuitivně odvodíme, že Jack má adekvátní příjem.

V klasické logice je lehké formalizovat tento případ odvozování zdravým rozumem, ale není jasné ani to, jak formalizovat uvedená fakta logickým programem. Mohli bychom použít tuto „disjunktivní databázi" :

$$Employed(Jack,Stanford) \mid Employed(Jack,SRI) \leftarrow . \tag{7}$$

$$Adequate\_Income(x) \leftarrow Employed(x) .$$

V hlavách pravidel budeme používat symbolu $|$ místo v logice obvyklého symbolu $\vee$ pro disjunkci, protože je určitý rozdíl mezi použitím první disjunkce a použitím a použitím disjunkce $\vee$ v klasické logice. Je to podobné rozdílu mezi nekontrapositivní $\leftarrow$ a kontrapositivní klasickou implikací nebo rozdílu mezi *not* a $\neg$.

Obecně definujeme *rozšířenou disjunktivní databázi* jako množinu pravidel tvaru

$$L_1 \,|\, L_2 \,|\, \dots \,|\, L_k \leftarrow L_{k+1}, \dots, L_{k+m}, not(L_{m+1}), \dots, not(L_n) \qquad (8)$$

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

Definici odpovědních množin rozšíříme i pro rozšířené disjunktivní databáze.

Nechť ***P*** je rozšířená disjunktivní databáze bez proměnných, která neobsahuje *not*. Nechť *Lit* je množina základních literálů v jazyce ***P***. Odpovědní množina pro ***P*** je minimální podmnožina *S* množiny *Lit*, která splňuje následující podmínky

(i) Pro každé pravidlo

$$L_1 \mid L_2 \mid \dots \mid L_k \leftarrow L_{k+1}, \dots, L_m$$

programu $\boldsymbol{P}$ takové, že literály $L_{k+1}, L_{k+2}, \dots, L_m$ patří do S, potom pro některé $i \leq k$ je L_i je také prvkem S,

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

Uvažujme databázi (7) kde druhé pravidlo je nahrazeno svými základními instancemi. Potom (7) má dvě odpovědní množiny

$$\{Employed(Jack,Stanford), Adequate\_income(Jack)\}$$

a

$$\{Employed(Jack,SRI), Adequate\_income(Jack)\}.$$

Odpověď na dotaz může nyní záviset na tom, která odpovědní množina se vybere. Například odpověď na dotaz *Employed(Jack,Stanford)*

První odpovědní množiny je *yes* , ale podle druhé je *no* . Odpověď na dotaz *Adequate_income(Jack)}* je v obou případech *yes* .

Nyní nechť $\boldsymbol{P}^S$ je rozšířená disjunktivní databáze, která vznikne z $\boldsymbol{P}$ vynecháním

(i) každého pravidla, které v těle obsahuje formuli *not L* , kde $L \in S$,

(ii) každé formule *not L* v tělech ostatních pravidel.

Pak $\boldsymbol{P}^S$ neobsahuje *not* , takže odpovědní množiny jsou již definovány. Je-li S některá z nich, řekneme, že S je odpovědní množina pro $\boldsymbol{P}$.

Chceme-li použít tuto definici odpovědní množiny k nějaké disjunktivní databázi s proměnnými, nahradíme nejprve každé pravidlo jeho základními instancemi.

Například, když doplníme databázi (7) o předpoklad uzavřeného světa pro predikát *Employed*

$$\neg\, Employed(x,y) \leftarrow not\ Employed(x,y)\ .$$

Potom nová databáze má dvě odpovědní množiny

$\{Employed(Jack,Stanford), \neg\, Employed(Jack,SRI), Adequate\_income(Jack)\}$

a

$\{Employed(Jack,SRI), \neg\, Employed(Jack,Stanford), Adequate\_income(Jack)\}$

Pro ilustraci rozdílu mezi $|$ a $\vee$ uvažujme databázi

$$Q \leftarrow P. \qquad (9)$$

$$P \mid \neg P \leftarrow .$$

Na rozdíl od pravidla vyloučeného třetího v klasické logice, druhé pravidlo v (8) nelze vynechat beze změny významu databáze.

Poslední pravidlo v (8) vyjadřuje fakt, že o P je buď známo, že je pravdivé nebo nepravdivé. Každá odpovědní množina obsahující takové pravidlo obsahuje buď P nebo $\neg P$.

Databáze obsahující jen pravidlo $Q \leftarrow P$. Má jen prázdnou odpovědní množinu, ale (8) má dvě odpovědní množiny $\{P,Q\}$ $\{\neg P\}$.

Redukce rozšířených programů na Obecné programy

Nechť $\boldsymbol{P}$ je rozšířený program. Nejprve budeme definovat jeho positivní tvar $\boldsymbol{P}^+$, který je obecným programem.

Pro každý predikát P v jazyce $\boldsymbol{P}$ nechť P' je nový predikát stejné četnosti jako P. Atom $P'(\ldots.)$ budeme nazývat positivním tvarem negativního literálu $\neg P(\ldots.)$.

Každý pozitivní literál je svým positivním tvarem. Positivní tvar literálu L budeme označovat L^+. Positivní tvar P^+ programu P vznikne nahrazením každé klauzule jejím positivním tvarem

$$L_0^+ \leftarrow L_1^+, L_2^+, \ldots, L_m^+, \mathit{not}\ L^+_{m+1}, \mathit{not}\ L^+_{m+2}, \ldots, \mathit{not}\ L^+_n . \qquad (10)$$

Například positivní tvar $\boldsymbol{P}_1^+$ popisující koncové vrcholy orientovaného grafu je

$$Terminal'(x) \leftarrow Arc(x,y).$$
$$Terminal(x) \leftarrow \mathit{not}\ Terminal(x).$$

Je to obvyklá definice koncových vrcholů v jazyce obecných logických programů.

Pro každou podmnožinu $S \subseteq Lit$, nechť S^+ označuje množinu positivních tvarů literálů z S.

Věta 2 Konsistentní množina $S \subseteq Lit$ je odpovědní množinou rozšířeného programu P právě když S^+ je odpovědní množinou P^+.

Transformace $P \;>\!\!\rightarrow P^+$ redukuje rozšířené programy na obecné programy i když P^+ nijak nenaznačuje, že P' intuitivně představuje negaci P.

Důkaz Věty 2 je používá dvou lemmat.

Lemma 1 Pro každý bezesporný program, který neobsahuje *not* platí

$$\alpha(P^+) = \alpha(P)^+$$

Důkaz. Podle předpokladu je P bezesporný program, takže α(P) je množina literálů, kterou lze generovat použitím (základních instancí) pravidel P

$$L_0 \leftarrow L_1 , \; \dots , \; L_m$$

jako odvozovacích pravidel. Podobně $\alpha(P^+)$ je množina atomů, která je generována použitím odpovídajících positivních pravidel

$$L_0^+ \leftarrow L_1^+ , \ \dots , \ L_m^+$$

Je zřejmé, že atomy generované pomocí těchto positivních pravidel jsou právě positivní tvary literálů generovaných pomocí pravidel původního programu.

Lemma 2 Pro každý sporný program P, který neobsahuje *not*, $\alpha(\mathrm{P}^+)$ obsahuje dvojici atomů tvaru A, $(\neg A)^+$.

Důkaz. Uvažujme množinu S, kterou lze generovat pomocí pravidel programu P jako odvozovacích pravidel. Předpokládejme, že P je bezesporný.

Potom $S = \alpha(\mathrm{P})$, takže $\alpha(\mathrm{P})$ je bezesporná. To není možné, protože program P je sporný. Tedy S je sporná.

Nechť A, $\neg A$ jsou komplementární literály z S, generované pomocí pravidel programu P jako odvozovacích pravidel.

Použitím odpovídajících positivních pravidel, můžeme odvodit

$$A\ ,\ (\neg\ A)^{+}$$

Potom tyto atomy patří do $\alpha(\mathrm{P}^{+})$.

Důkaz Věty 2. Bez újmy na obecnosti můžeme předpokládat, že P neobsahuje proměnné. Nechť S je bezesporná množina literálů. Podle definice je S^{+} odpovědní množinou pro P když

$$S^{+} = \alpha(\mathrm{P}^{+(S+)}) \ ,$$

Což lze přepsat jako $S^{+} = \alpha((\mathrm{P}^{S})^{+})$.

Nyní je třeba dokázat ekvivalenci

$$S^{+} = \alpha((\mathrm{P}^{S})^{+}) \leftrightarrow S = \alpha((\mathrm{P}^{S})) \qquad (11)$$

Případ 1. P^{S} je bezesporný.

Potom podle Lemmatu 2 je levá strana (11) ekvivalentní rovnosti

$$S^{+} = \alpha(\mathrm{P}^{S})^{+}$$

a to je ekvivalentní pravé straně (10).

Případ 2. P^{S} je sporný. Protože S je podle předpokladu bezesporná množina, pravá strana (10) je nepravdivá.. Podle lemmatu 2 množina $\alpha((\mathrm{P}^{S})^{+})$ obsahuje dvojici $A, \neg A$. Protože S je bezesporná, S^{+} nemůže obsahovat takovou dvojici literálů, takže levá strana (11) je také nepravdivá.

Důsledek. Je-li množina $S \subseteq Lit$ bezesporná a S^{+} je jediná odpovědní množina pro P^{+}, potom S je jediná odpovědní množina pro P.

Důkaz. Podle Věty 2 je S odpovědní množinou pro P, je tedy třeba jen ukázat že P nemá žádné jiné odpovědní množiny. Mějme nějakou množinu $S' \subseteq Lit$, která by byla odpovědní množinou pro P.

Protože S je bezesporná, P je bezesporný podle lemmatu 2 , takže S' je také bezesporná množina. Podle Věty 2 je $(S')^+$ je odpovědní množinou pro P^+ , takže $(S')^+ = S^+$ odkud plyne $S' = S$.

Poznámka. Tento důsledek ukazuje jak se dají použít procedury pro vyhodnocení dotazů v obecných programech pro (některé) rozšířené programy.

Má-li program P^+ „dobré“ vlastnosti (například je-li stratifikovaný) a jeho odpovědní množin neobsahuje dvojici atomů A , $(\neg A)^+$, potom program P má také „dobré“ vlastnosti (v tomto případě je stratifiko-vaný) a literál $L \in Lit$ je prvkem odpovědní množiny pro P , právě když je základní atom L^+ prvkem odpovědní množiny pro P^+ .

Podmínka $L^+ \in \mathrm{P}^+$. se dá v principu ověřit obvyklým systémem logického programování.

Například, že literály z množiny (5) patří do odpovědní množiny pro program P_1 lze v Prologu ověřit vyhodnocením (tří) dotazů pro P_1^+

$$FairGPA(Ann)\ ,\ (HighGPA)'(Ann),\quad Interview(Ann)$$

Dotazy obsahující proměnné lze zpracovat obdobným způsobem.

Poznámka. Předpoklad bezespornosti množiny S ve Větě 2 a jejím důsledku je nutný i v případě, že P je bezesporný a P^+ je stratifikovaný.

Ukazuje to tento příklad programu P_2 :

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

Tento program nemá žádnou odpovědní množinu i když P_2^+ je stratifikovaný program. Odpovědní množina pro P_2^+ je positivní tvar sporné množiny $\{P, Q,, \neg Q\}$.