Rezoluční metoda ve výrokové logice s aplikací na logické programování

Vít Fojtík

1 Úvod

V informatice je v mnoha případech potřeba řešit problémy, které se zakládají na přesných pravidlech, ze kterých je třeba něco vyvodit, nebo ověřit. Takové úlohy se často objevují v oblastech jako je automatické dokazování, počítačová lingvistika, umělá inteligence, automatické plánování, nebo databázové systémy. V těchto případech je možné použít metody běžného procedurálního programování, ale to může vést k těžkopádným a málo přizpůsobitelným programům. Efektivně se pro řešení těchto problémů dá použít logické programování.

Logické programování používá predikátovou logiku jako základ svých programů. V této práci se omezím na jeden z nejrozšířenějších jazyků logického programování, Prolog. Jeho programy se skládají z pravidel a faktů. Pravidla mají obecně tvar:

H :- B_1, …, B_n.

kde H, B_1, …, B_n jsou predikátové atomy. Takové pravidlo lze interpretovat dvěma způsoby, buď deklarativně: „Pokud platí B_1 a … a B_n, pak platí H," nebo procedurálně: „Pokud vyřešíme problémy B_1, …, B_n, pak jsme vyřešili problém H". Například:

Savec(X) :- Kocka(X)

může být použito jako procedura pro ověření, že nějaké X je savec, tak, že ověříme, že X je kočka, nebo jako procedura pro nalezení X, které je savec, nalezením X, které je kočka.

Fakta jsou v Prologu tvaru:

H.

Po formulaci pravidel a faktů můžeme programu klást dotazy typu „Splňuje A formuli P(X)?" nebo „Která X splňují P(X)?", na které program vypíše odpověď. Prolog je deklarativní jazyk, tj. programátor neřídí běh programu, jen určí cíle výpočtu a za průběh ručí interpret jazyka.

Nyní pro ilustraci uvedu jednoduchý příklad programu v Prologu z [1], který ukazuje základní možnosti dotazování:

```
has(jack,apples).
has(ann,plums).
has(dan,money).
fruit(apples).
fruit(plums).

| ?- has(jack,apples).                    /* does Jack have apples? */
yes.
| ?- has(X,plums).                        /* who has plums? */
X = ann
| ?- has(X,apples),has(X,plums).          /* does anyone have apples and plums? */
no.
| ?- has(X,Y),not fruit(Y).               /* does anyone have something else? */
X = dan
Y = money
```

V různých implementacích Prologu se může algoritmus pro vyhodnocení dotazů lišit, ale všechny používají SLD rezoluci. SLD rezoluce je logické odvozovací pravidlo, které využívá Hornovy formule a je speciálním případem rezoluční metody.

2 Rezoluce

Rezoluční metoda je odvozovací pravidlo, použitelné jako důkazový systém ve výrokové i predikátové logice (v této práci se omezím pouze na výrokový případ). Protože rezoluce předpokládá formule v konjunktivně normálním tvaru (CNF), nejprve zadefinuji potřebné pojmy.

Definice: Formuli l nazýváme literálem, pokud je výrokovým atomem, nebo jeho negací (tj. $l = p \vee l = \neg p,\ p \in At$). (Disjunktivní) klauzulí rozumíme disjunkci literálů (díky asociativitě a komutativitě disjunkce nezáleží u klauzule na pořadí literálů). Formule je v konjunktivně normálním tvaru, pokud jde o konjunkci klauzulí.

Věta (bez důkazu): Pro každou formuli existuje ekvivalentní formule v CNF.

Rezoluční metoda se používá pro ukázání spornosti (obvykle konečné – ale ne nutně) množiny klauzulí. Takovou konečnou množinu je možné ekvivalentně chápat jako konjunkci jejích prvků, tedy formuli v konjunktivně normálním tvaru.

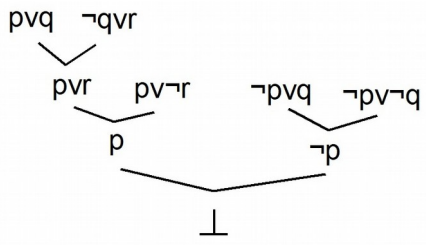
Rezoluční pravidlo má následující tvar (z [4]):

$$\frac{a_1 \vee \ldots \vee a_{i-1} \vee c \vee a_{i+1} \vee \ldots \vee a_n, \quad b_1 \vee \ldots \vee b_{j-1} \vee \neg c \vee b_{j+1} \vee \ldots \vee b_m}{a_1 \vee \ldots \vee a_{i-1} \vee a_{i+1} \vee \ldots \vee a_n \vee b_1 \vee \ldots \vee b_{j-1} \vee b_{j+1} \vee \ldots \vee b_m}$$

kde a_i a b_j jsou literály a c je atom.Tedy pokud nějaká klauzule obsahuje opačný literál k literálu obsaženému v jiné klauzuli, lze odvodit spojení těchto klauzulí s vynecháním daných literálů. Výslednou formuli nazýváme rezolventou předchozích formulí. Speciálním případem jsou pak dvě jednoprvkové klauzule:

$$\frac{p, \neg p}{\bot}$$

kde z atomu a jeho negace lze odvodit spor. Při použití rezoluce pro dokazování formulí se cílová formule zneguje, převede do CNF a poté se opakovaným použitím rezolučního pravidla dojde ke sporu. Takový rezoluční kalkul je korektní i úplný. Pro ilustraci uvedu rezoluční důkaz formule $(\neg p \wedge \neg q) \vee (q \wedge \neg r) \vee (\neg p \wedge r) \vee (p \wedge \neg q) \vee (p \wedge q)$, jejíž negace je $(p \vee q) \wedge (\neg q \vee r) \wedge (p \vee \neg r) \wedge (\neg p \vee q) \wedge (\neg p \vee \neg q)$:



Pro přesnost definuji rezoluční důkaz i formálně.

Definice: Rezoluční důkaz klauzule C z množiny klauzulí T je posloupnost $C_0, \ldots, C_n$, kde $C_n = C$ a pro každé $i \leq n$ je C_i prvkem množiny T, nebo rezolventou C_j a C_k pro nějaké $j,k < i$.

Věta (bez důkazu): Pro klauzuli C a množinu klauzulí T platí: $T \models C$, právě když existuje rezoluční důkaz C z T.

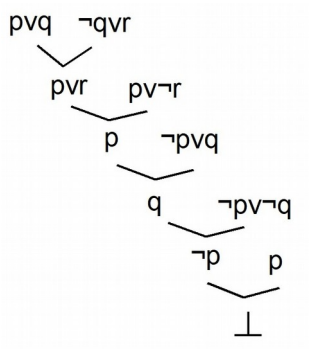
3 Varianty rezoluce a Hornovy formule

Nyní postupně vybuduji zvláštní případy rezoluce až po SLD rezoluci.

Definice: Důkaz klauzule C z množiny klauzulí T lineární rezolucí je posloupnost dvojic $(C_0, B_0), \ldots, (C_{n-1}, B_{n-1})$, kde C je rezolventou C_{n-1} a B_{n-1}, C_0 je prvkem T, pro každé $i > 0$ je C_i rezolventou C_{i-1}

a B_{i-1} a pro každé i je B_i buď rovno C_j pro nějaké j < i, nebo je prvkem T.

Lineární rezoluce tedy nařizuje, aby vždy jedna z formulí, na které aplikujeme rezoluční pravidlo, byla buď z množiny předpokladů T, nebo aby byla jednou z předchozích rezolvent. Tím se zakazuje stromovité větvení rezolučních důkazů. Každý důkaz lineární rezoluce lze převést na důkaz obyčejné rezoluce a naopak, takže lineární rezoluce je opět korektní a úplná. Například lineární verze předchozího důkazu by mohla vypadat takto:



Věta (bez důkazu): Pro klauzuli C a množinu klauzulí T existuje rezoluční důkaz C z T právě když existuje důkaz C z T lineární rezolucí.

V dalším kroku budeme požadovat, aby jedna z formulí byla přímo z množiny předpokladů (tedy zakážeme, aby byla nějakou předchozí rezolventou).

Definice: Důkaz klauzule C z množiny klauzulí T LI rezolucí je posloupnost dvojic (C_0, B_0), …, (C_{n-1}, B_{n-1}), kde rezolventou C_{n-1} a B_{n-1} je ⊥, $C_0 = C$, pro každé i > 0 je C_i rezolventou C_{i-1} a B_{i-1} a pro každé i je B_i prvkem T. Klauzuli C_i nazýváme i-tý cíl a B_i i-tá boční formule.

LI rezoluce sice je korektní, ale již není úplná. Je úplná jen pro tzv. Hornovy formule.

Definice: Klauzuli nazveme Hornovou, pokud obsahuje nejvýše jeden pozitivní literál. Hornova formule je konjunkce Hornových klauzulí.

Pravidlo je Hornova klauzule obsahující právě jeden pozitivní literál a alespoň jeden negativní. Faktem nazýváme jednoprvkovou (Hornovu) klauzuli obsahující pouze pozitivní literál. Cíl je Hornova klauzule, která neobshuje pozitivní literál.

Věta (bez důkazu): Pro C cíl a T množinu Hornových klauzulí platí: pokud $T \models C$, existuje důkaz C z T LI rezolucí.

Nyní si můžeme všimnout, že programům v Prologu odpovídají Hornovy formule složené jen z pravidel a faktů (takovým formulím budeme říkat programy). Například pravidlo

$$H :- B_1, \ldots, B_n.$$

můžeme interpretovat jako formuli

$$(b_1 \wedge \ldots \wedge b_n) \rightarrow h,$$

která je z vlastností implikace ekvivalentní

$$\neg(b_1 \wedge \ldots \wedge b_n) \vee h,$$

což je dle De Morganových zákonů ekvivalentní Hornovu pravidlu

$$\neg b_1 \vee \ldots \vee \neg b_n \vee h.$$

Zjevně lze na Hornova fakta převést fakta Prologu. Cíle potom v Prologu odpovídají dotazům. Dotazy mají na výrokové úrovni obecně formu

$$?- p_1, \ldots, p_n,$$

čímž vlastně požadujeme, aby Prolog dokázal formuli

$$p_1 \wedge \ldots \wedge p_n$$

*(*z formulí daného programu). Po vzoru dokazování rezoluční metodou se formule zneguje na cíl

$$\neg p_1 \vee \ldots \vee \neg p_n,$$

a pokud se z ní podaří odvodit spor, Prolog vrátí *yes.* Nyní prozkoumáme, jak se případný spor odvozuje. Následující varianty rezoluce se již nebudou myšlenkou příliš lišit od LI rezoluce, ale pro algoritmické použití je třeba přesně specifikovat technické detaily.

Zatím jsme sice díky asociativitě a komutativitě disjunkce nebrali v úvahu pořadí literálů v klauzuli, ale v počítači se klauzule reprezentují jako uspořádané seznamy literálů, takže je nutné rezoluční pravidlo poupravit.

Klauzule programu jsou pouze pravidla a fakta, tedy každá obsahuje právě jeden pozitivní literál. Bývá zvykem tento literál řadit jako první v seznamu. Potom má pravidlo obecně tvar

$$(p_i, \neg q_1, ..., \neg q_n).$$

Věta: Pokud je C cíl a T program, pak v důkazu C z T LI rezolucí je cíl každého kroku Hornovým cílem.
Důkaz: Indukcí dle i dokážu tvrzení pro i-tý cíl. Pro i = 0 plyne tvrzení z předpokladu. Pokud je C_{i-1} cíl a B_{i-1} fakt nebo pravidlo, pak všechny jejich literály jsou negativní, krom prvního literálu B_{i-1}. Tento literál se tedy musí použít k rezoluci a neobjeví se v rezolventě C_i, která tudíž bude mít všechny literály negativní.

Předchozího tvrzení využívá následující definice.

Definice: Důkaz LD rezolucí je důkaz LI rezolucí, ve kterém v každém kroku rezolventa aktuálního cíle ($\neg p_1$, ..., $\neg p_{i-1}$, $\neg p_i$, $\neg p_{i+1}$, ..., $\neg p_n$) a boční klauzule (p_i, $\neg q_1$, ..., $\neg q_m$) je ($\neg p_1$, ..., $\neg p_{i-1}$, $\neg q_1$, ..., $\neg q_m$, $\neg p_{i+1}$, ..., $\neg p_n$).

LD rezoluce je tedy varianta LI rezoluce, která specifikuje pořadí literálů v rezolventě. Další zbývající záležitostí je volba literálu z aktuálního cíle, přes který se bude v dalším kroku rezolvovat. V mnoha případech se volí první literál seznamu, ale některé interprety Prologu používají jiná selekční pravidla, která umožňují pozměnit průběh rezolučního algoritmu.

Definice: Selekční pravidlo je libovolné zobrazení R, které každému cíli C přiřadí přirozené číslo R(C), které je menší než počet literálů v C. (Nejčastější pravidlo je dáno předpisem R(C) = 0 pro libovolný cíl C)

Definice: Důkaz SLD rezolucí dle selekčního pravidla R je důkaz LD rezolucí, který v i-tém kroku rezolvuje přes R(C_i)-tý literál klauzule C_i.

Věta (bez důkazu): Pro C klauzuli a P program platí: existuje důkaz C z P LI rezolucí, právě když existuje důkaz C z P SLD rezolucí.

Z předchozí věty plyne, že SLD rezoluce je úplná pro dotazy na programy v Prologu.

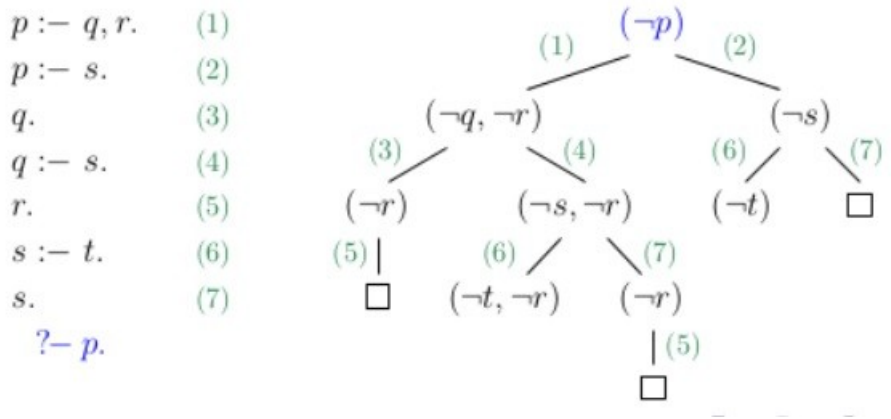
4 SLD strom

Poslední nevyřešenou součástí algoritmu je volba boční klauzule. V i-tém kroku se určí literál, přes který se bude rezolvovat, ale v programu se může nacházet více pravidel a faktů, které daný literál obshují. Možnosti, jak tuto klauzuli zvolit, generují implicitní strom zvaný SLD strom, který definuji induktivně.

Definice: SLD strom pro počáteční cíl C, program P a selekční pravidlo R je definován:

1. V kořeni je cíl C.
2. Pokud C_i je klauzule ve vrcholu stromu a B je klauzule z P, jejíž pozitivní literál je opačným literálem k R(C_i)-tému literálu klauzule C_i, pak daný vrchol má syna obsahujícího rezolventu C_i a B.
3. (Neexistují jiné vrcholy, než vzniklé pravidly 1. a 2.)

Skutečný výpočet tedy spočívá v procházení SLD stromu, buď dokud se nenajde vrchol obsahující spor (pak Prolog vrátí *yes.*), nebo dokud není prohledán celý strom (Prolog vrátí *no.*). Následující příklad z [3] ukzuje vlevo program s dotazem a vpravo SLD strom pro selekční pravidlo R(C) = 0. (Pro spor se zde užívá symbol □)



Jednotlivé interprety se liší ve způsobu prohledávání SLD stromu. Protože ale strom nemusí být konečný (kvůli cyklickým pravidlům), nezachovávají některé způsoby prohledávání (např. prohledávání do hloubky) úplnost, což ilustruje následující příklad z [3]:

5 Rezoluční algoritmus

Na závěr této práce uvedu příklad schématu konkrétního algoritmu pro SLD rezoluci, který používá pravidlo R(C) = 0 a prohledává SLD strom do šířky.

Algoritmus: (C je zadaný cíl, P program, Rezolventa(C_1, C_2) je funkce dle definice LD rezoluce, F fronta)

1. Pro každé B z P proveď: {
2. Pokud B[0] = C[0], zařaď na konec F výsledek Rezolventa(C, B); }
3. Dokud F není prázdná {
4. Ze začátku F odeber G;
5. Pokud je G prázdná (spor), vypiš ”*yes.*” a ukonči ptogram;
6. Jinak pro každé B z P proveď: {
7. Pokud B[0] = G[0], zařaď na konec F výsledek Rezolventa(C, B); } }
8. Vypiš ”*no.*”

Takový algoritmus je sice jednoduchý, ale kvůli nahodilé volbě bočních klauzulí bude neefektivní.

6 Zdroje

[1] M. Malita – Examples in Prolog http://www.anselm.edu/homepage/mmalita/culpro/index.html
[2] Wikipedia – Prolog https://en.wikipedia.org/wiki/Prolog
[3] P. Gregor – Prezentace k přednášce Výroková a predikátová logika
http://ktiml.mff.cuni.cz/~gregor/logika2015/index.html
[4] Wikipedia – Resolution https://en.wikipedia.org/wiki/Resolution_(logic)