Principy logického programování

v Prologu

Šárka Vavrečková

Ústav informatiky, Filozoficko-přírodovědecká fakulta Slezské univerzity v Opavě

sarka.vavreckova@fpf.slu.cz

7. listopadu 2008

Vlastnosti Prologu

Prolog

- je deklarativní – určujeme, co se má provést, a ne jak konkrétně se to má provést, procedurální část výpočtu je čistě v režii Prologu,
- je založen na rekurzi – rekurzívně prohledává lineární výpočetní strom přes jednotlivé uzly do hloubky,
- nerozlišuje data a program, obojí je v jednom souboru ve stejných klauzulích,

Vlastnosti Prologu

Prolog

- je deklarativní – určujeme, co se má provést, a ne jak konkrétně se to má provést, procedurální část výpočtu je čistě v režii Prologu,
- je založen na rekurzi – rekurzívně prohledává lineární výpočetní strom přes jednotlivé uzly do hloubky,
- nerozlišuje data a program, obojí je v jednom souboru ve stejných klauzulích,

Vlastnosti Prologu

Prolog

- je deklarativní – určujeme, co se má provést, a ne jak konkrétně se to má provést, procedurální část výpočtu je čistě v režii Prologu,
- je založen na rekurzi – rekurzívně prohledává lineární výpočetní strom přes jednotlivé uzly do hloubky,
- nerozlišuje data a program, obojí je v jednom souboru ve stejných klauzulích,

Vlastnosti Prologu

Prolog

- globální proměnné neexistují, lokální existují v rámci jediné klauzule, pokud jsou stejně pojmenovány proměnné v různých klauzulích, Prolog je rozliší přidáním čísla klauzule k názvu proměnné – místo `X` bude používat v klauzuli číslo 25 proměnnou `X25` apod.,
- si pamatuje, které klauzule použil při předchozích použitích rezoluce a jaké unifikace zvolil,
- pro unifikaci používá algoritmus podobný hledání nejobecnějšího unifikátoru.

Vlastnosti Prologu

Prolog

- globální proměnné neexistují, lokální existují v rámci jediné klauzule, pokud jsou stejně pojmenovány proměnné v různých klauzulích, Prolog je rozliší přidáním čísla klauzule k názvu proměnné – místo `X` bude používat v klauzuli číslo 25 proměnnou `X25` apod.,
- si pamatuje, které klauzule použil při předchozích použitích rezoluce a jaké unifikace zvolil,
- pro unifikaci používá algoritmus podobný hledání nejobecnějšího unifikátoru.

Vlastnosti Prologu

Prolog

- globální proměnné neexistují, lokální existují v rámci jediné klauzule, pokud jsou stejně pojmenovány proměnné v různých klauzulích, Prolog je rozliší přidáním čísla klauzule k názvu proměnné – místo `X` bude používat v klauzuli číslo 25 proměnnou `X25` apod.,
- si pamatuje, které klauzule použil při předchozích použitích rezoluce a jaké unifikace zvolil,
- pro unifikaci používá algoritmus podobný hledání nejobecnějšího unifikátoru.

Zpracování dotazů

Slovně:

Platí A_1 a zároveň A_2 a zároveň . . . ?

Zpracování dotazů

Slovně:

Platí A_1 a zároveň A_2 a zároveň . . . ?

Co se dá dosadit

za proměnné X, Y, Kdo, atd. použité v dotazu?

Zpracování dotazů

Slovně:

Platí A_1 a zároveň A_2 a zároveň . . . ?

Existuje něco, co se dá dosadit

za proměnné X, Y, Kdo, atd. použité v dotazu?

Zpracování dotazů

Slovně:

Platí A_1 a zároveň A_2 a zároveň . . . ?

Existuje něco, co se dá dosadit

za proměnné X, Y, Kdo, atd. použité v dotazu?

$$
\begin{array}{lll}
 & \neg\exists u_1 \exists u_2 \ldots \exists u_r & (A_1 \,\&\, A_2 \,\&\, \ldots \,\&\, A_p) \\
\Leftrightarrow & \forall u_1 \forall u_2 \ldots \forall u_r & (\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_p) \\
\Leftrightarrow & \forall u_1 \forall u_2 \ldots \forall u_r & (true \;\rightarrow\; (\neg A_1 \vee \neg A_2 \vee \ldots \vee \neg A_p)) \\
\Leftrightarrow & & A_1, A_2, \ldots, A_p \rightarrow
\end{array}
$$

Pojmy

Cílová klauzule

= klauzule, na kterou chceme použít rezoluci, hledáme další klauzuli, která se dá s cílovou klauzulí unifikovat. První cílová klauzule je negovaný dotaz.
Cílové klauzule mají vždy prázdný konsekvent (hlavu).

Cíle

= atomy (tj. predikáty včetně argumentů) cílové klauzule, nejdůležitější je cíl v klauzuli nejvíc vlevo.

Příklad

V následujícím příkladu bude v každé cílové klauzuli zakroužkován hlavní cíl, který je právě vyhodnocován (hledáme klauzuli, s jejíž hlavou (konsekventem) se dá tento cíl unifikovat).

Pojmy

Cílová klauzule

= klauzule, na kterou chceme použít rezoluci, hledáme další klauzuli, která se dá s cílovou klauzulí unifikovat. První cílová klauzule je negovaný dotaz.
Cílové klauzule mají vždy prázdný konsekvent (hlavu).

Cíle

= atomy (tj. predikáty včetně argumentů) cílové klauzule, nejdůležitější je cíl v klauzuli nejvíc vlevo.

Příklad

V následujícím příkladu bude v každé cílové klauzuli zakroužkován hlavní cíl, který je právě vyhodnocován (hledáme klauzuli, s jejíž hlavou (konsekventem) se dá tento cíl unifikovat).

Pojmy

Cílová klauzule

= klauzule, na kterou chceme použít rezoluci, hledáme další klauzuli, která se dá s cílovou klauzulí unifikovat. První cílová klauzule je negovaný dotaz.
Cílové klauzule mají vždy prázdný konsekvent (hlavu).

Cíle

= atomy (tj. predikáty včetně argumentů) cílové klauzule, nejdůležitější je cíl v klauzuli nejvíc vlevo.

Příklad

V následujícím příkladu bude v každé cílové klauzuli zakroužkován hlavní cíl, který je právě vyhodnocován (hledáme klauzuli, s jejíž hlavou (konsekventem) se dá tento cíl unifikovat).

Příklad

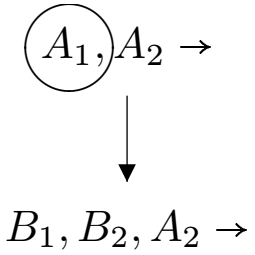
$A_1, A_2 \rightarrow$

1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$

```
1. A2 .
2. A3 .
3. B2 .
4. B3 .
5. B1 :- A3 .
6. A3 :- B3 .
7. A1 :- B1 , B2 .
```

Příklad

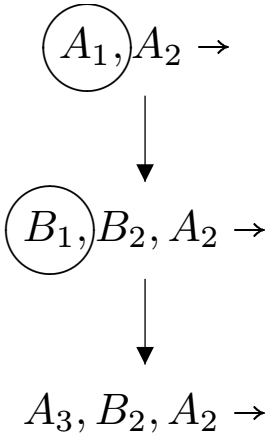
1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$




```
1. A2 .
2. A3 .
3. B2 .
4. B3 .
5. B1 :- A3 .
6. A3 :- B3 .
7. A1 :- B1 , B2 .
```

Příklad

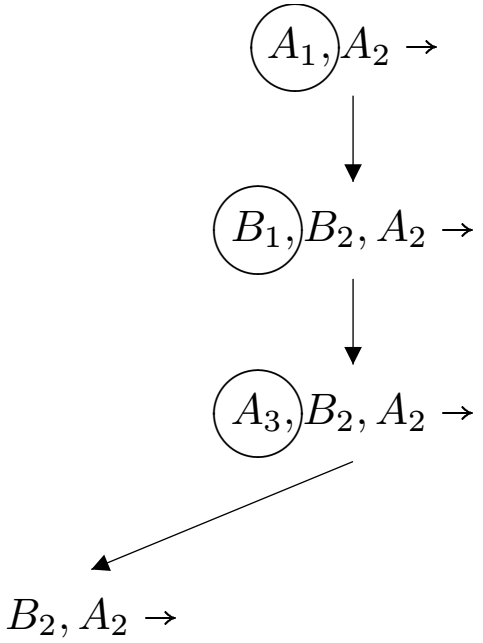
1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$




```
1. A2 .
2. A3 .
3. B2 .
4. B3 .
5. B1 :- A3 .
6. A3 :- B3 .
7. A1 :- B1 , B2 .
```

Příklad

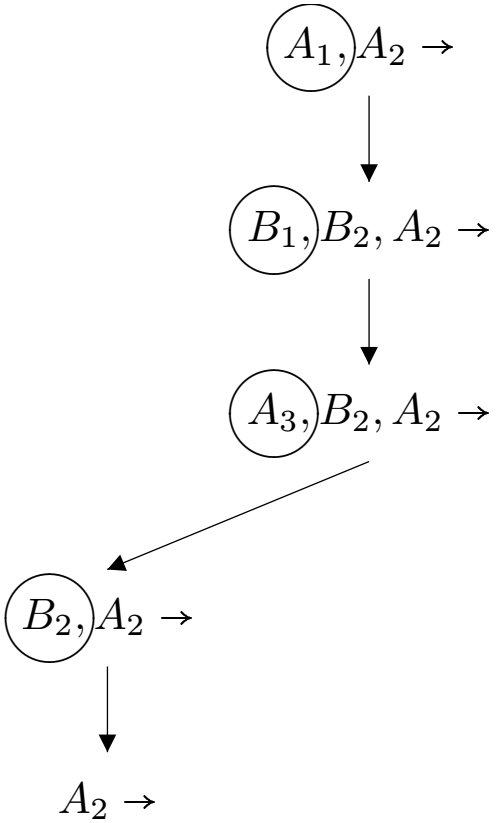
1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$




```
1. A2 .
2. A3 .
3. B2 .
4. B3 .
5. B1 :- A3 .
6. A3 :- B3 .
7. A1 :- B1 , B2 .
```

Příklad

1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$

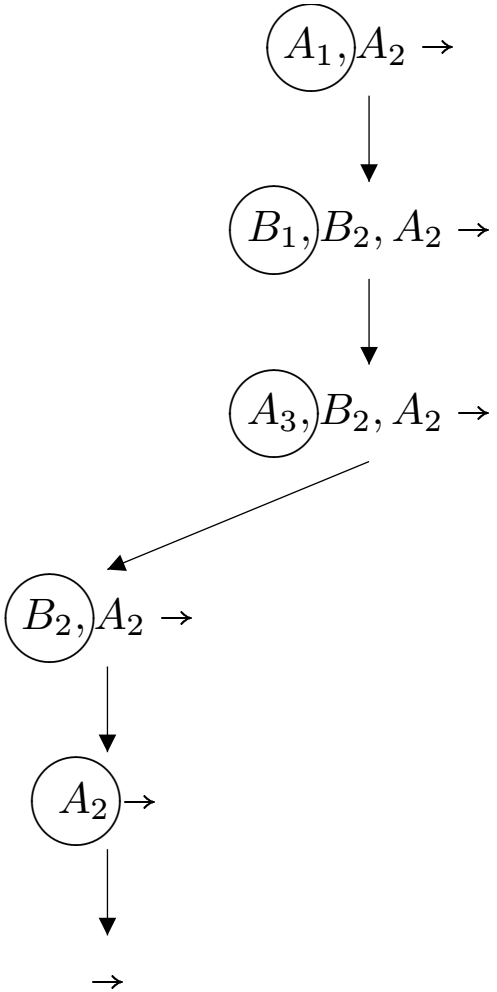


1. `A2.`
2. `A3.`
3. `B2.`
4. `B3.`
5. `B1 :- A3.`
6. `A3 :- B3.`
7. `A1 :- B1, B2.`

Příklad

1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$

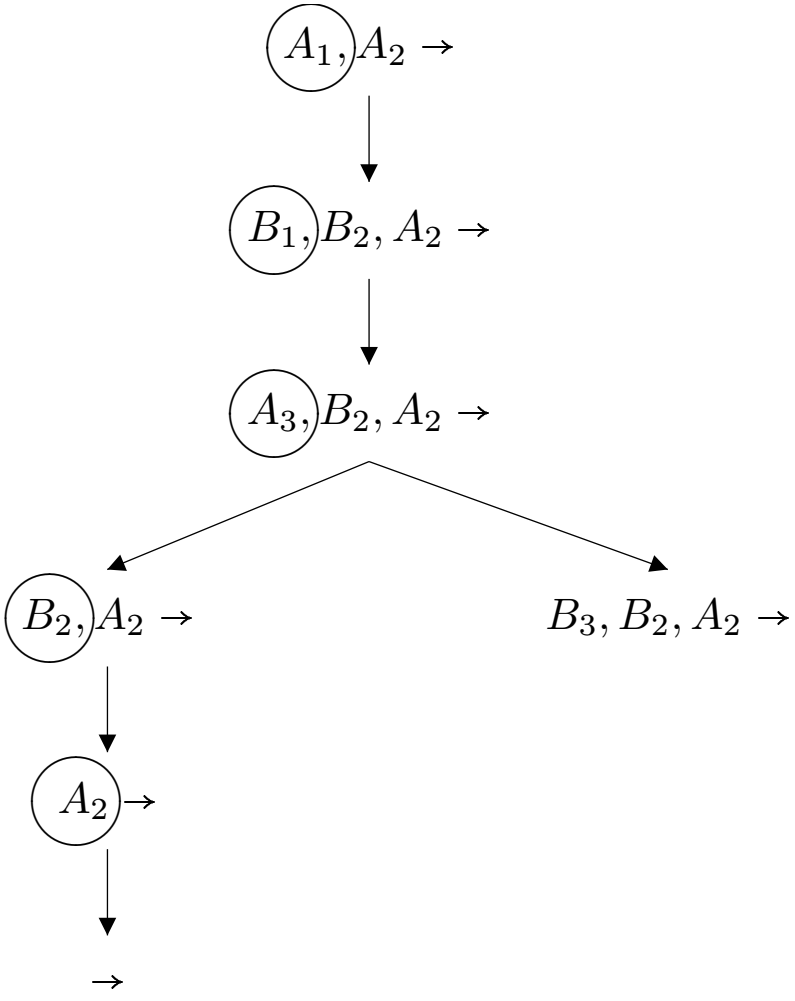
1. `A2.`
2. `A3.`
3. `B2.`
4. `B3.`
5. `B1 :- A3.`
6. `A3 :- B3.`
7. `A1 :- B1, B2.`



Příklad

1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$

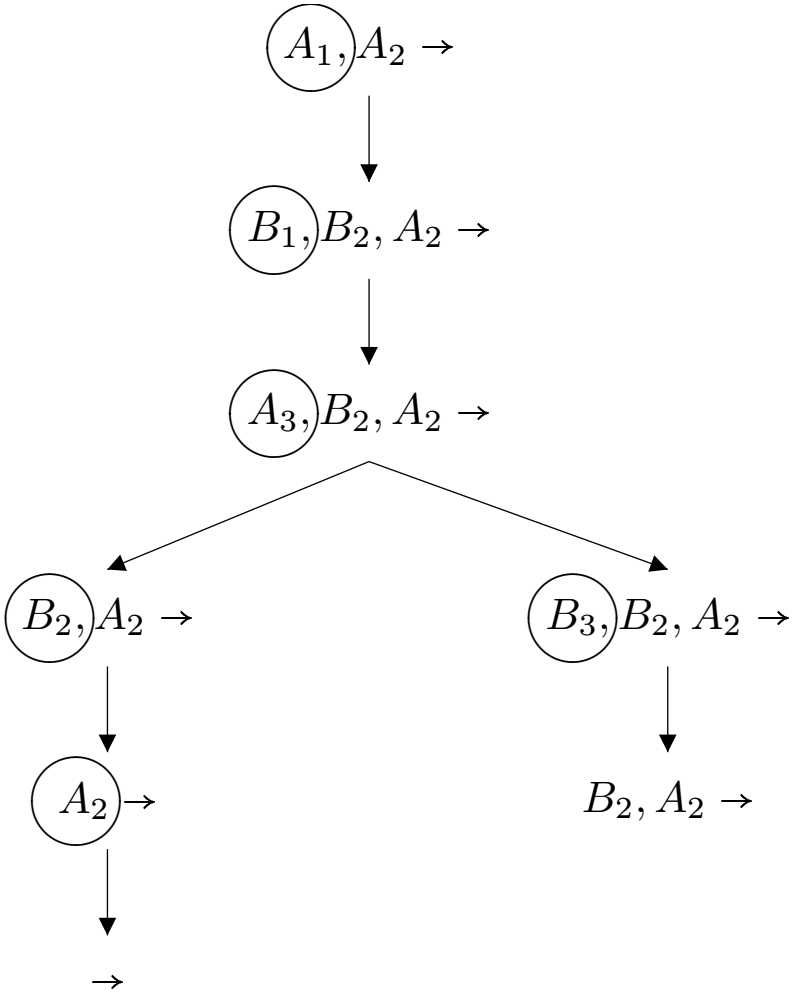
```
1. a2.
2. a3.
3. b2.
4. b3.
5. b1 :- a3.
6. a3 :- b3.
7. a1 :- b1, b2.
```



Příklad

1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$

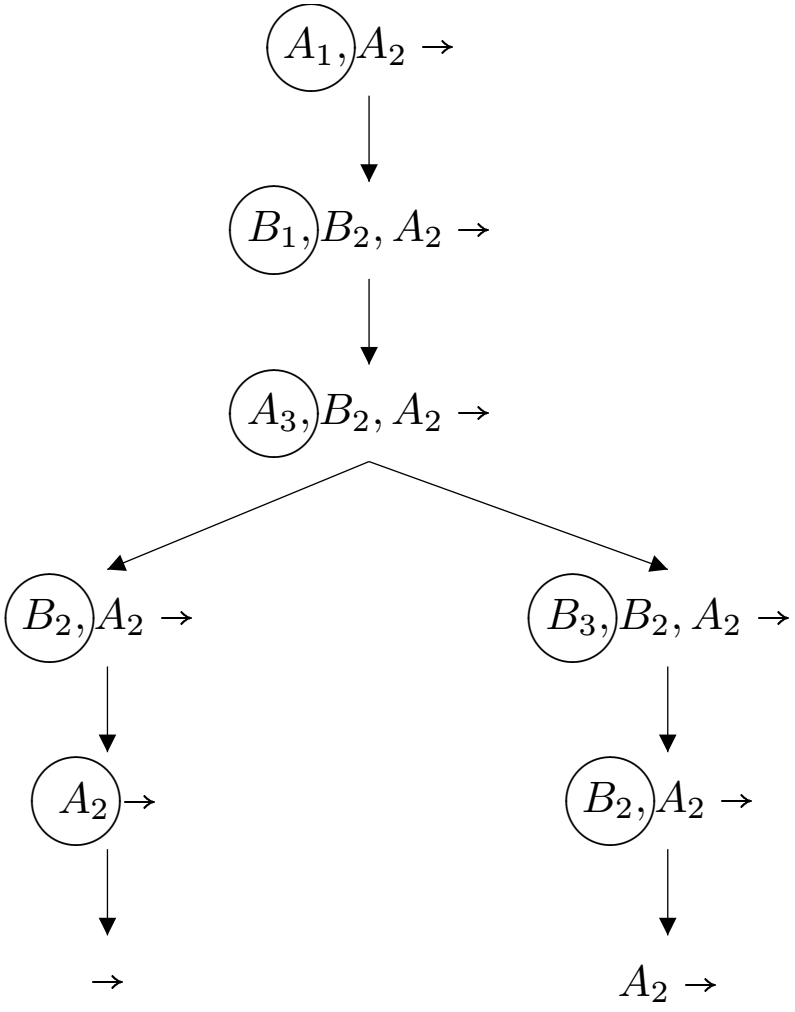
```
1. A2 .
2. A3 .
3. B2 .
4. B3 .
5. B1 :- A3 .
6. A3 :- B3 .
7. A1 :- B1 , B2 .
```



Příklad

1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$

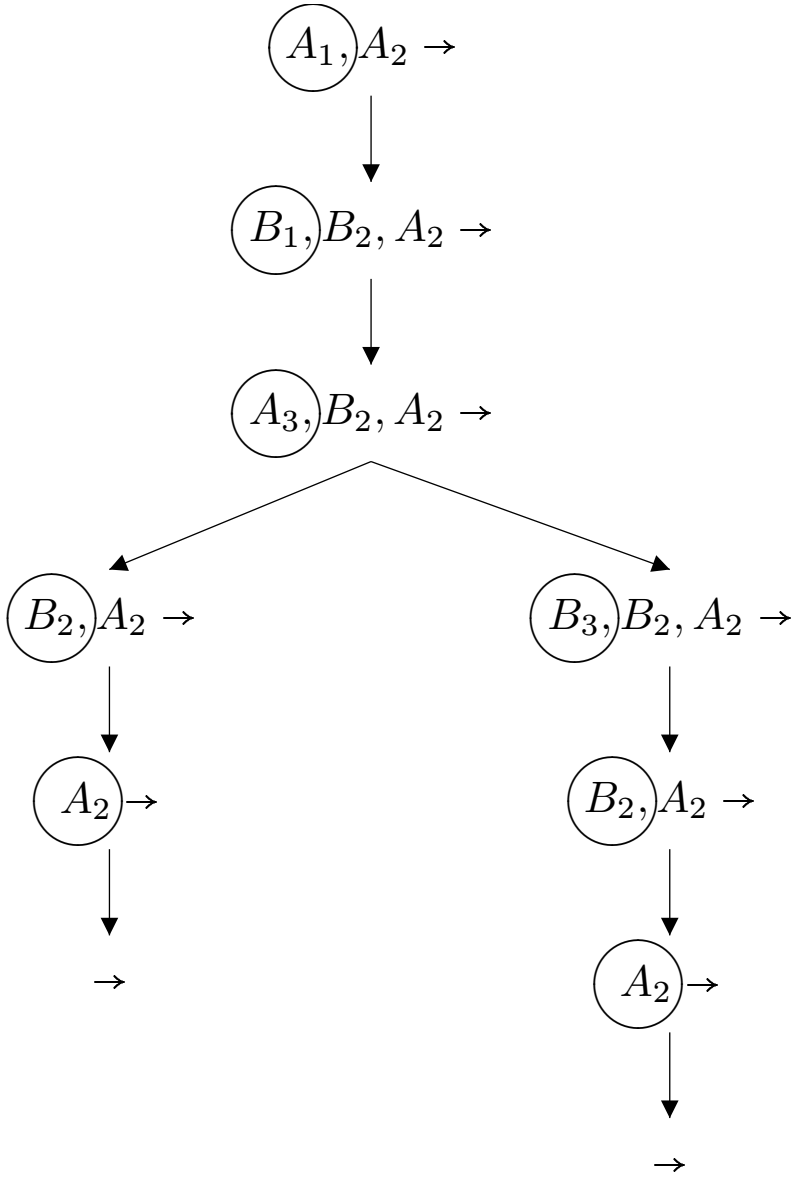
```
1. A2 .
2. A3 .
3. B2 .
4. B3 .
5. B1 :- A3 .
6. A3 :- B3 .
7. A1 :- B1 , B2 .
```



Příklad

1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$

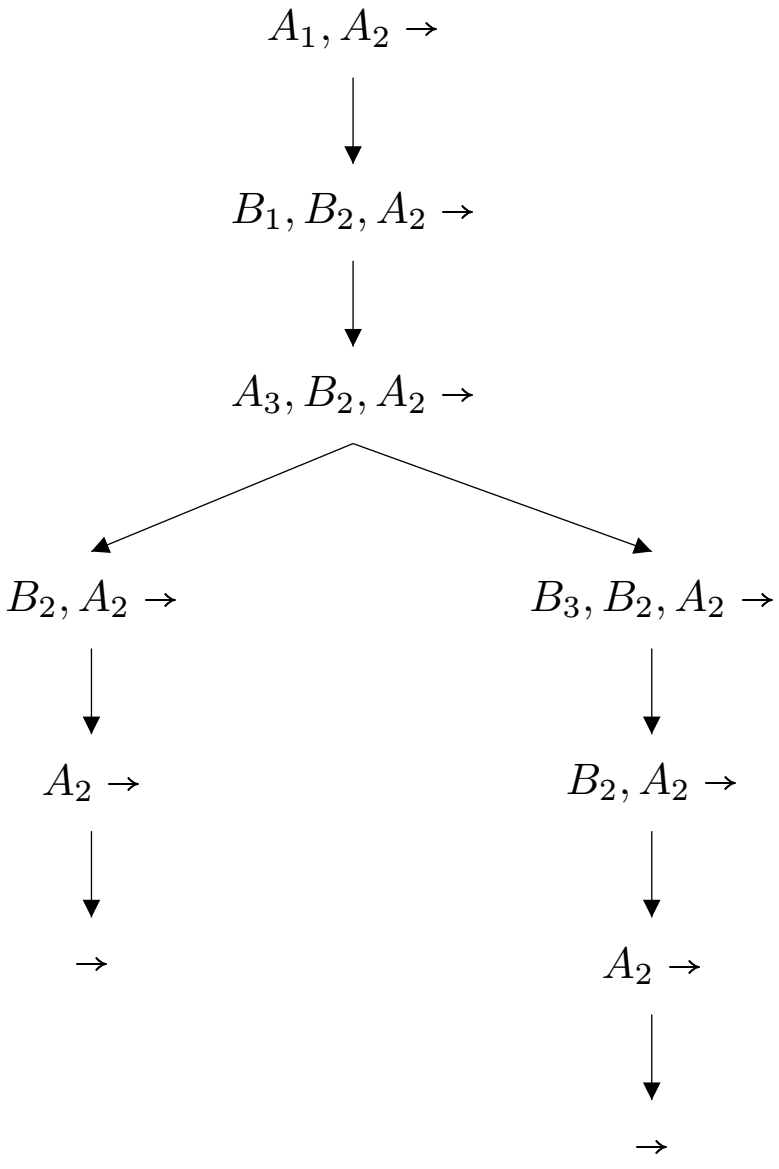
```
1. A2 .
2. A3 .
3. B2 .
4. B3 .
5. B1 :- A3 .
6. A3 :- B3 .
7. A1 :- B1 , B2 .
```



Příklad

1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$

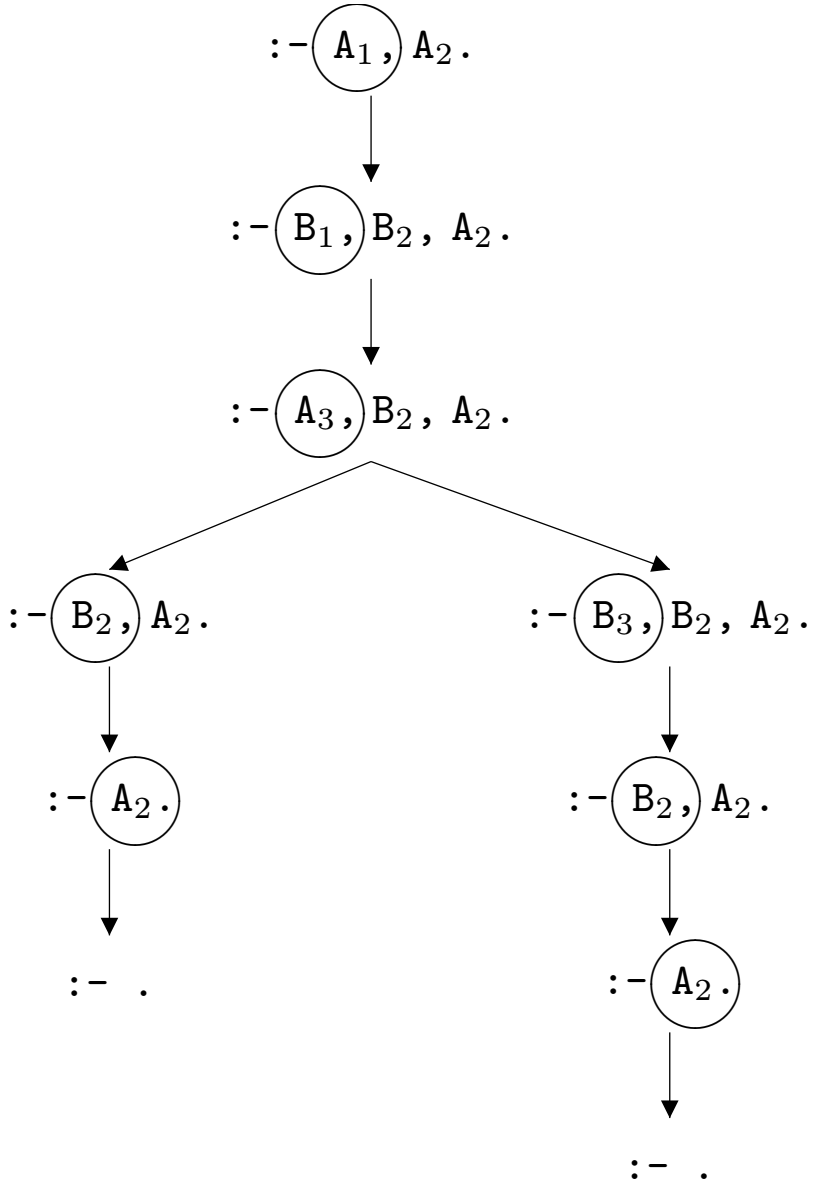
```
1. A2 .
2. A3 .
3. B2 .
4. B3 .
5. B1 :- A3 .
6. A3 :- B3 .
7. A1 :- B1 , B2 .
```



Příklad

1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$

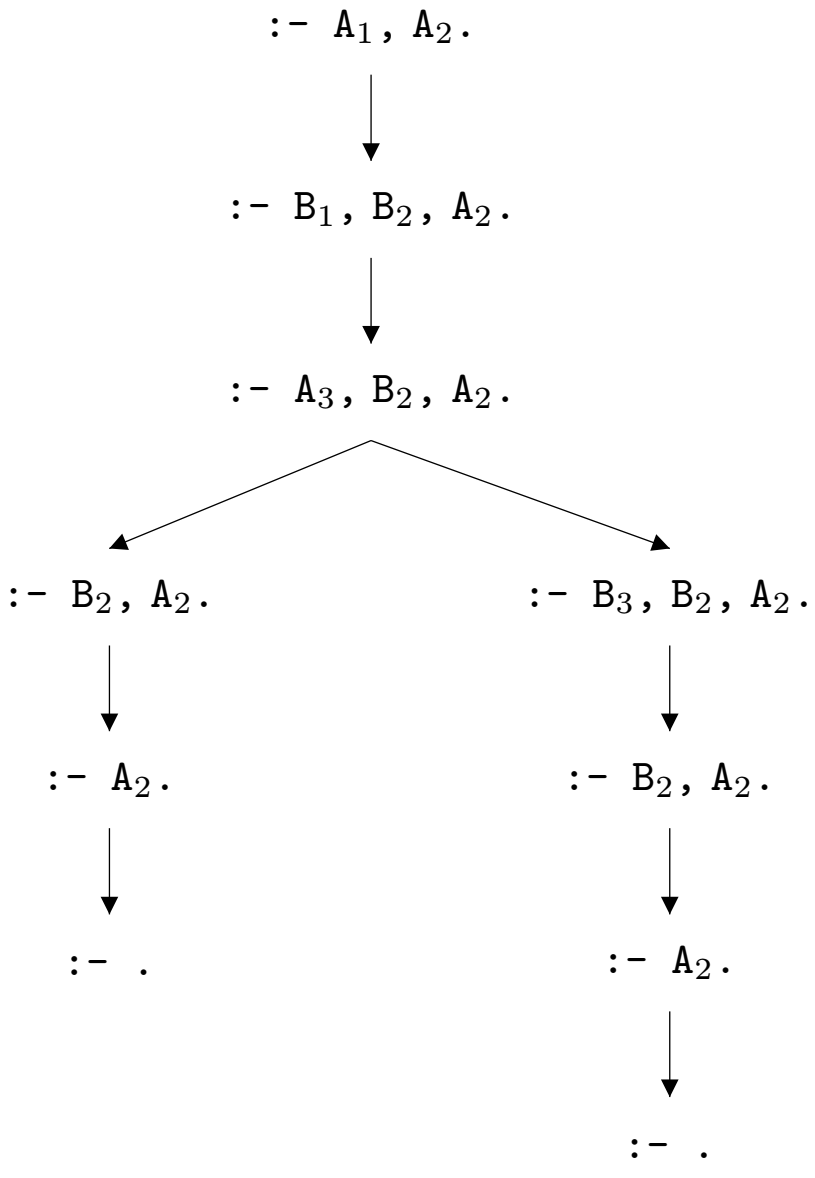
1. `A2.`
2. `A3.`
3. `B2.`
4. `B3.`
5. `B1 :- A3.`
6. `A3 :- B3.`
7. `A1 :- B1, B2.`



Příklad

1. $\rightarrow A_2$
2. $\rightarrow A_3$
3. $\rightarrow B_2$
4. $\rightarrow B_3$
5. $A_3 \rightarrow B_1$
6. $B_3 \rightarrow A_3$
7. $B_1, B_2 \rightarrow A_1$

1. `A2.`
2. `A3.`
3. `B2.`
4. `B3.`
5. `B1 :- A3.`
6. `A3 :- B3.`
7. `A1 :- B1, B2.`



Proměnné a zásobník

Přidáváme proměnné

Výpočetní strom dotazu se větví:

- když existuje více klauzulí, jejichž hlava se dá unifikovat s hlavním cílem,
- když existuje více možností dosazení hodnot za proměnné v hlavním cíli.

Ve skutečnosti druhá možnost souvisí s první, viz předchozí příklad.

Pokud jsou v dotazu (univerzálně kvantifikované) proměnné (`X`, `Prom`, `Kdo`, `...`), musíme si pamatovat jejich hodnotu:

- pro tutéž proměnnou je v každé větvi výpočetního stromu jiná hodnota,
- je třeba si ji pamatovat až k listu (konci) větve,
- když se dostaneme k listu větve, vypíšeme všechny zapamatované hodnoty proměnných z dotazu.

Proměnné a zásobník

Přidáváme proměnné

Výpočetní strom dotazu se větví:

- když existuje více klauzulí, jejichž hlava se dá unifikovat s hlavním cílem,
- když existuje více možností dosazení hodnot za proměnné v hlavním cíli.

Ve skutečnosti druhá možnost souvisí s první, viz předchozí příklad. Pokud jsou v dotazu (univerzálně kvantifikované) proměnné (`X`, `Prom`, `Kdo`, `...`), musíme si pamatovat jejich hodnotu:

- pro tutéž proměnnou je v každé větvi výpočetního stromu jiná hodnota,
- je třeba si ji pamatovat až k listu (konci) větve,
- když se dostaneme k listu větve, vypíšeme všechny zapamatované hodnoty proměnných z dotazu.

Proměnné a zásobník

Ukládání do zásobníku

Zásobník je paměťová struktura typu zásobník (Last In, First Out), do které ukládáme údaje o pohybu ve výpočetním stromě.

Každý prvek zásobníku obsahuje údaje použité při vytvoření nového uzlu výpočetního stromu (unifikace + rezoluce):

- číslo klauzule, na kterou je zároveň s cílovou klauzulí uplatněna v daném kroku rezoluce,
- unifikace (co je za které proměnné dosazeno).

Proměnné a zásobník

Ukládání do zásobníku

Zásobník je paměťová struktura typu zásobník (Last In, First Out), do které ukládáme údaje o pohybu ve výpočetním stromě.
Každý prvek zásobníku obsahuje údaje použité při vytvoření nového uzlu výpočetního stromu (unifikace + rezoluce):

- číslo klauzule, na kterou je zároveň s cílovou klauzulí uplatněna v daném kroku rezoluce,
- unifikace (co je za které proměnné dosazeno).

Proměnné a zásobník

Vyjímání ze zásobníku

se provádí na konci každé větve (ať už úspěšné nebo neúspěšné). Vyjmeme uspořádanou dvojici $[i, \varphi]$ – když byl údaj ukládán, byla použita unifikace φ na klauzuli číslo i a cílovou klauzuli, víme také, které predikáty klauzulí byly použity.

Proměnné a zásobník

Vyjímání ze zásobníku

Po ukončení jedné větve začneme vytvářet další:

- jestliže byla větev ukončena úspěchem, vypíšeme hodnoty požadovaných proměnných,
- vyjmeme údaj ze zásobníku a provedeme *backtracking* (navracení) – rekonstruujeme původní cílovou klauzuli podle údaje ze zásobníku, tím se ve stromě posuneme o uzel výše,
- uzel níže, ze kterého jsme přišli, vznikl unifikací cílové klauzule a klauzule číslo i, proto hledáme klauzuli pro unifikaci od klauzule číslo $i + 1$,
- když najdeme vhodnou klauzuli pro unifikaci, tvoříme rekurzívně další větev, když ne, ukončíme výpočet s výsledkem `no`.

Proměnné a zásobník

Vyjímání ze zásobníku

Po ukončení jedné větve začneme vytvářet další:

- jestliže byla větev ukončena úspěchem, vypíšeme hodnoty požadovaných proměnných,
- vyjmeme údaj ze zásobníku a provedeme *backtracking* (navracení) – rekonstruujeme původní cílovou klauzuli podle údaje ze zásobníku, tím se ve stromě posuneme o uzel výše,
- uzel níže, ze kterého jsme přišli, vznikl unifikací cílové klauzule a klauzule číslo i, proto hledáme klauzuli pro unifikaci od klauzule číslo $i + 1$,
- když najdeme vhodnou klauzuli pro unifikaci, tvoříme rekurzívně další větev, když ne, ukončíme výpočet s výsledkem `no`.

Proměnné a zásobník

Vyjímání ze zásobníku

Po ukončení jedné větve začneme vytvářet další:

- jestliže byla větev ukončena úspěchem, vypíšeme hodnoty požadovaných proměnných,
- vyjmeme údaj ze zásobníku a provedeme *backtracking* (navracení) – rekonstruujeme původní cílovou klauzuli podle údaje ze zásobníku, tím se ve stromě posuneme o uzel výše,
- uzel níže, ze kterého jsme přišli, vznikl unifikací cílové klauzule a klauzule číslo i, proto hledáme klauzuli pro unifikaci od klauzule číslo $i + 1$,
- když najdeme vhodnou klauzuli pro unifikaci, tvoříme rekurzívně další větev, když ne, ukončíme výpočet s výsledkem `no`.

Proměnné a zásobník

Vyjímání ze zásobníku

Po ukončení jedné větve začneme vytvářet další:

- jestliže byla větev ukončena úspěchem, vypíšeme hodnoty požadovaných proměnných,
- vyjmeme údaj ze zásobníku a provedeme *backtracking* (navracení) – rekonstruujeme původní cílovou klauzuli podle údaje ze zásobníku, tím se ve stromě posuneme o uzel výše,
- uzel níže, ze kterého jsme přišli, vznikl unifikací cílové klauzule a klauzule číslo i, proto hledáme klauzuli pro unifikaci od klauzule číslo $i + 1$,
- když najdeme vhodnou klauzuli pro unifikaci, tvoříme rekurzívně další větev, když ne, ukončíme výpočet s výsledkem `no`.

Algoritmus

1. Na začátku výpočtu se cílovou klauzulí stane (negovaný) dotaz.
2. Jestliže je cílovou klauzulí prázdná klauzule, *končíme výpočet větve s úspěchem* (`yes`):
   - 2.1 V dotazu jsou proměnné: vypíšeme nalezené hodnoty těchto proměnných (poslední hodnoty ze zásobníku příslušející těmto proměnným), čekáme na stisk klávesy a pokud je to klávesa ;, provedeme navracení a pokračujeme bodem 3.
   - 2.2 V dotazu nejsou proměnné: vypíšeme `yes`, ukončíme celý výpočet a smažeme obsah zásobníku.

Algoritmus

1. Na začátku výpočtu se cílovou klauzulí stane (negovaný) dotaz.
2. Jestliže je cílovou klauzulí prázdná klauzule, *končíme výpočet větve s úspěchem* (`yes`):
   - 2.1 V dotazu jsou proměnné: vypíšeme nalezené hodnoty těchto proměnných (poslední hodnoty ze zásobníku příslušející těmto proměnným), čekáme na stisk klávesy a pokud je to klávesa `;`, provedeme navracení a pokračujeme bodem 3.
   - 2.2 V dotazu nejsou proměnné: vypíšeme `yes`, ukončíme celý výpočet a smažeme obsah zásobníku.

Algoritmus

1. Na začátku výpočtu se cílovou klauzulí stane (negovaný) dotaz.
2. Jestliže je cílovou klauzulí prázdná klauzule, *končíme výpočet větve s úspěchem* (`yes`):
   - 2.1 V dotazu jsou proměnné: vypíšeme nalezené hodnoty těchto proměnných (poslední hodnoty ze zásobníku příslušející těmto proměnným), čekáme na stisk klávesy a pokud je to klávesa `;`, provedeme navracení a pokračujeme bodem 3.
   - 2.2 V dotazu nejsou proměnné: vypíšeme `yes`, ukončíme celý výpočet a smažeme obsah zásobníku.

Algoritmus

1. Na začátku výpočtu se cílovou klauzulí stane (negovaný) dotaz.
2. Jestliže je cílovou klauzulí prázdná klauzule, *končíme výpočet větve s úspěchem* (`yes`):
   - 2.1 V dotazu jsou proměnné: vypíšeme nalezené hodnoty těchto proměnných (poslední hodnoty ze zásobníku příslušející těmto proměnným), čekáme na stisk klávesy a pokud je to klávesa ;, provedeme navracení a pokračujeme bodem 3.
   - 2.2 V dotazu nejsou proměnné: vypíšeme `yes`, ukončíme celý výpočet a smažeme obsah zásobníku.

Algoritmus

3. Vezmeme nejlevější atom (cíl) cílové klauzule a hledáme v programu klauzuli, která
   - ještě pro tento cíl nebyla použita,
   - má ve své hlavě (tj. v konsekventu) tentýž predikát jako testovaný cíl
   - a je možné provést unifikaci přes atom v hlavě klauzule a testovaný cíl cílové klauzule.

   Jsou tři možnosti:

   3.1 Takovou klauzuli najdeme: pokračujeme bodem 4.

   3.2 Takovou klauzuli se nepodaří najít a zásobník není prázdný: provedeme navracení (vše, co bylo až do této pozice v programu provedeno, zrušíme a zkoušíme další cestu) a pokračujeme bodem 3.

   3.3 Takovou klauzuli se nepodaří najít a zásobník je prázdný: nelze pokračovat jinak, než jak se dosud postupovalo (tj. zásobník je prázdný, ale cílová klauzule je neprázdná), *končíme výpočet s neúspěchem* (vypíšeme `no`).

Algoritmus

3. Vezmeme nejlevější atom (cíl) cílové klauzule a hledáme v programu klauzuli, která
   - ještě pro tento cíl nebyla použita,
   - má ve své hlavě (tj. v konsekventu) tentýž predikát jako testovaný cíl
   - a je možné provést unifikaci přes atom v hlavě klauzule a testovaný cíl cílové klauzule.

   Jsou tři možnosti:

   3.1 Takovou klauzuli najdeme: pokračujeme bodem 4.

   3.2 Takovou klauzuli se nepodaří najít a zásobník není prázdný: provedeme navracení (vše, co bylo až do této pozice v programu provedeno, zrušíme a zkoušíme další cestu) a pokračujeme bodem 3.

   3.3 Takovou klauzuli se nepodaří najít a zásobník je prázdný: nelze pokračovat jinak, než jak se dosud postupovalo (tj. zásobník je prázdný, ale cílová klauzule je neprázdná), *končíme výpočet s neúspěchem* (vypíšeme `no`).

Algoritmus

3. Vezmeme nejlevější atom (cíl) cílové klauzule a hledáme v programu klauzuli, která
   - ještě pro tento cíl nebyla použita,
   - má ve své hlavě (tj. v konsekventu) tentýž predikát jako testovaný cíl
   - a je možné provést unifikaci přes atom v hlavě klauzule a testovaný cíl cílové klauzule.

   Jsou tři možnosti:

   3.1 Takovou klauzuli najdeme: pokračujeme bodem 4.

   3.2 Takovou klauzuli se nepodaří najít a zásobník není prázdný: provedeme navracení (vše, co bylo až do této pozice v programu provedeno, zrušíme a zkoušíme další cestu) a pokračujeme bodem 3.

   3.3 Takovou klauzuli se nepodaří najít a zásobník je prázdný: nelze pokračovat jinak, než jak se dosud postupovalo (tj. zásobník je prázdný, ale cílová klauzule je neprázdná), *končíme výpočet s neúspěchem* (vypíšeme `no`).

Algoritmus

3. Vezmeme nejlevější atom (cíl) cílové klauzule a hledáme v programu klauzuli, která
   - ještě pro tento cíl nebyla použita,
   - má ve své hlavě (tj. v konsekventu) tentýž predikát jako testovaný cíl
   - a je možné provést unifikaci přes atom v hlavě klauzule a testovaný cíl cílové klauzule.

   Jsou tři možnosti:

   3.1 Takovou klauzuli najdeme: pokračujeme bodem 4.

   3.2 Takovou klauzuli se nepodaří najít a zásobník není prázdný: provedeme navracení (vše, co bylo až do této pozice v programu provedeno, zrušíme a zkoušíme další cestu) a pokračujeme bodem 3.

   3.3 Takovou klauzuli se nepodaří najít a zásobník je prázdný: nelze pokračovat jinak, než jak se dosud postupovalo (tj. zásobník je prázdný, ale cílová klauzule je neprázdná), *končíme výpočet s neúspěchem* (vypíšeme `no`).

Algoritmus

3. Vezmeme nejlevější atom (cíl) cílové klauzule a hledáme v programu klauzuli, která
   - ještě pro tento cíl nebyla použita,
   - má ve své hlavě (tj. v konsekventu) tentýž predikát jako testovaný cíl
   - a je možné provést unifikaci přes atom v hlavě klauzule a testovaný cíl cílové klauzule.

   Jsou tři možnosti:

   3.1 Takovou klauzuli najdeme: pokračujeme bodem 4.

   3.2 Takovou klauzuli se nepodaří najít a zásobník není prázdný: provedeme navracení (vše, co bylo až do této pozice v programu provedeno, zrušíme a zkoušíme další cestu) a pokračujeme bodem 3.

   3.3 Takovou klauzuli se nepodaří najít a zásobník je prázdný: nelze pokračovat jinak, než jak se dosud postupovalo (tj. zásobník je prázdný, ale cílová klauzule je neprázdná), *končíme výpočet s neúspěchem* (vypíšeme `no`).

Algoritmus

3. Vezmeme nejlevější atom (cíl) cílové klauzule a hledáme v programu klauzuli, která
   - ještě pro tento cíl nebyla použita,
   - má ve své hlavě (tj. v konsekventu) tentýž predikát jako testovaný cíl
   - a je možné provést unifikaci přes atom v hlavě klauzule a testovaný cíl cílové klauzule.

   Jsou tři možnosti:

   3.1 Takovou klauzuli najdeme: pokračujeme bodem 4.

   3.2 Takovou klauzuli se nepodaří najít a zásobník není prázdný: provedeme navracení (vše, co bylo až do této pozice v programu provedeno, zrušíme a zkoušíme další cestu) a pokračujeme bodem 3.

   3.3 Takovou klauzuli se nepodaří najít a zásobník je prázdný: nelze pokračovat jinak, než jak se dosud postupovalo (tj. zásobník je prázdný, ale cílová klauzule je neprázdná), *končíme výpočet s neúspěchem* (vypíšeme `no`).

Algoritmus

3. Vezmeme nejlevější atom (cíl) cílové klauzule a hledáme v programu klauzuli, která
   - ještě pro tento cíl nebyla použita,
   - má ve své hlavě (tj. v konsekventu) tentýž predikát jako testovaný cíl
   - a je možné provést unifikaci přes atom v hlavě klauzule a testovaný cíl cílové klauzule.

   Jsou tři možnosti:

   3.1 Takovou klauzuli najdeme: pokračujeme bodem 4.

   3.2 Takovou klauzuli se nepodaří najít a zásobník není prázdný: provedeme navracení (vše, co bylo až do této pozice v programu provedeno, zrušíme a zkoušíme další cestu) a pokračujeme bodem 3.

   3.3 Takovou klauzuli se nepodaří najít a zásobník je prázdný: nelze pokračovat jinak, než jak se dosud postupovalo (tj. zásobník je prázdný, ale cílová klauzule je neprázdná), *končíme výpočet s neúspěchem* (vypíšeme `no`).

Algoritmus

4. Unifikujeme cílovou klauzuli a nalezenou klauzuli, uplatníme pravidlo rezoluce a rezolventu (výsledek rezoluce) použijeme jako *novou cílovou klauzuli*. Je zřejmé, že nejlevější cíl, který jsme zpracovávali v původní cílové klauzuli, se v nové cílové klauzuli neobjeví.
   Do zásobníku je uloženo číslo klauzule, která je unifikována s cílem, a údaje o použité substituci. Pokračujeme bodem 2.

Příklad použití algoritmu

viz skripta, str. 114.

Algoritmus

4. Unifikujeme cílovou klauzuli a nalezenou klauzuli, uplatníme pravidlo rezoluce a rezolventu (výsledek rezoluce) použijeme jako *novou cílovou klauzuli*. Je zřejmé, že nejlevější cíl, který jsme zpracovávali v původní cílové klauzuli, se v nové cílové klauzuli neobjeví.
   Do zásobníku je uloženo číslo klauzule, která je unifikována s cílem, a údaje o použité substituci. Pokračujeme bodem 2.

Příklad použití algoritmu

viz skripta, str. 114.

Vestavěné predikáty pro řízení výpočtu

Predikát `call/1`

pouze vyhodnotí svůj argument a vrátí jeho výsledek.

Predikát nepravdy – `fail`

Predikát `fail` je vždy vyhodnocen jako $false$, slouží k okamžitému zastavení vyhodnocování větve s neúspěchem, spustí vyhledávání další větve.

Predikát řezu – !

Predikát ! je sice vyhodnocen jako $true$ (tj. pustí při vyhodnocování dále do větve výpočetního stromu), ale znemožní navracení (backtracking), tedy všechna další řešení jsou odříznuta, znemožní hledání dalších větví výpočtu.
Slouží především k odříznutí „nechtěných“ dalších řešení nebo nevhodné další rekurze (nejsou vyhledávány další klauzule, jejichž hlava se dá unifikovat s cílem těsně před !).

Vestavěné predikáty pro řízení výpočtu

Predikát `call/1`

pouze vyhodnotí svůj argument a vrátí jeho výsledek.

Predikát nepravdy – `fail`

Predikát `fail` je vždy vyhodnocen jako $false$, slouží k okamžitému zastavení vyhodnocování větve s neúspěchem, spustí vyhledávání další větve.

Predikát řezu – !

Predikát ! je sice vyhodnocen jako $true$ (tj. pustí při vyhodnocování dále do větve výpočetního stromu), ale znemožní navracení (backtracking), tedy všechna další řešení jsou odříznuta, znemožní hledání dalších větví výpočtu.
Slouží především k odříznutí „nechtěných“ dalších řešení nebo nevhodné další rekurze (nejsou vyhledávány další klauzule, jejichž hlava se dá unifikovat s cílem těsně před !).

Vestavěné predikáty pro řízení výpočtu

Predikát `call/1`

pouze vyhodnotí svůj argument a vrátí jeho výsledek.

Predikát nepravdy – `fail`

Predikát `fail` je vždy vyhodnocen jako $false$, slouží k okamžitému zastavení vyhodnocování větve s neúspěchem, spustí vyhledávání další větve.

Predikát řezu – !

Predikát ! je sice vyhodnocen jako $true$ (tj. pustí při vyhodnocování dále do větve výpočetního stromu), ale znemožní navracení (backtracking), tedy všechna další řešení jsou odříznuta, znemožní hledání dalších větví výpočtu.
Slouží především k odříznutí „nechtěných“ dalších řešení nebo nevhodné další rekurze (nejsou vyhledávány další klauzule, jejichž hlava se dá unifikovat s cílem těsně před !).

Příklad

Definice predikátu `not`

```
not(X) :- call(X),!,fail.
not(X).
```

Podobný princip – vyloučení

```
pritomen(X) :- X=honza,!,fail.
pritomen(X).
```

Příklad

Definice predikátu not

```
not(X) :- call(X),!,fail.
not(X).
```

Podobný princip – vyloučení

```
pritomen(X) :- X=honza,!,fail.
pritomen(X).
```

Zajímavé odkazy

Test Zone – on-line editor Prologu

`http://kti.mff.cuni.cz/ bartak/prolog/testing.html`

Expertní systémy v Prologu

`http://www.amzi.com/ExpertSystemsInProlog/`