1 Rezoluce v predikátovém počtu

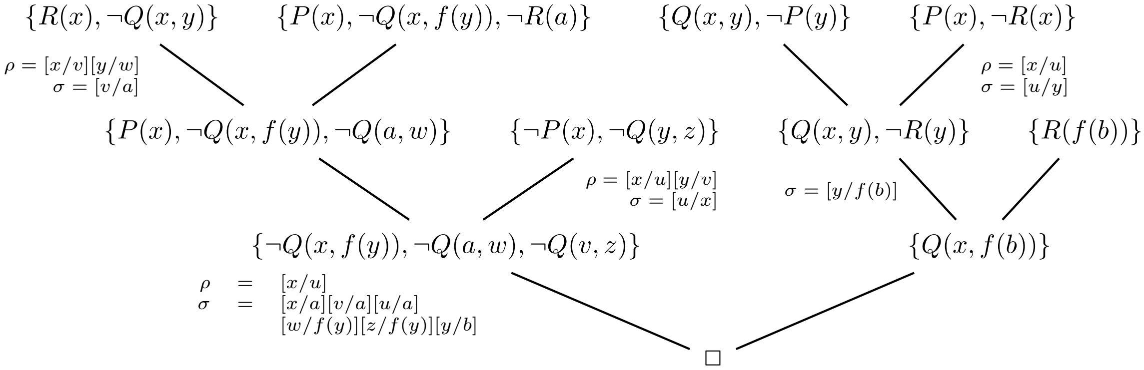
Příklad 1.1: Vyvraťte následující množinu klauzulí

$$\begin{aligned} S \;=\; & \{\{P(x), \neg Q(x, f(y)), \neg R(a)\}, \{R(x), \neg Q(x, y)\}, \{\neg P(x), \neg Q(y, z)\}, \\ & \{P(x), \neg R(x)\}, \{R(f(b))\}, \{Q(x, y), \neg P(y)\}\} \end{aligned}$$

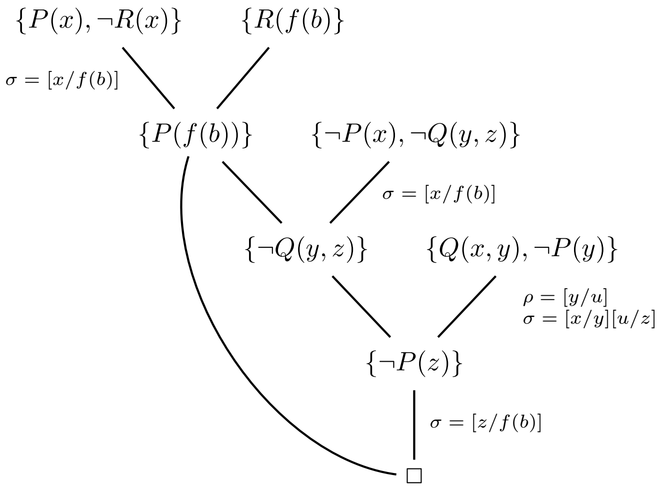
pomocí obecné rezoluce, lineární rezoluce, LI rezoluce, LD rezoluce a SLD rezoluce[1].

Řešení 1.1:

a) Obecná rezoluce



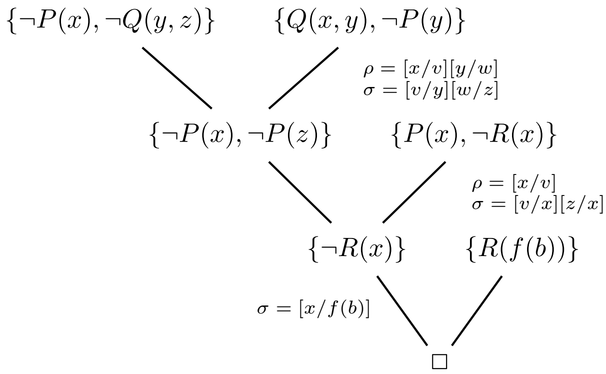
b) Lineární rezoluce – rezolvujeme vždy předchozí rezolventu s klauzulí z vyvracené množiny nebo dříve odvozenou rezolventou. Důkaz má lineární strukturu. Lineární rezoluce je úplná.



c) LI rezoluce (lineární vstupní rezoluce) – lineární vstupní rezoluce začíná cílovou klauzulí (klauzulí, která neobsahuje žádný pozitivní literál) a rezolvuje vždy předchozí rezolventou s klauzulí z vyvracené množiny. Jedná se

[1] Viz materiály k předmětu *IB101 Úvod do logiky a logického programování* (sedmá přednáška).

o zjemnění lineární rezoluce, které není obecně úplné. Aplikací na množinu Hornových klauzulí dostaneme úplnou metodu.



d) LD rezoluce – zjemnění LI rezoluce, které pracuje výhradně s množinou Hornových klauzulí. Klauzule jsou nahrazeny uspořádanými seznamy stejně jako ve výrokové logice. Z původní množiny S obdržíme množinu

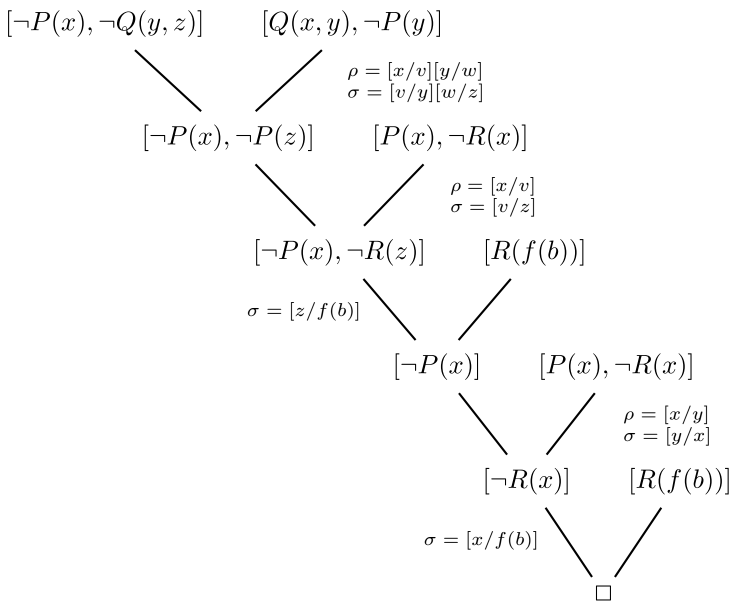
$$\begin{aligned} S' &= \{[P(x), \neg Q(x, f(y)), \neg R(a)], [R(x), \neg Q(x,y)], [\neg P(x), \neg Q(y,z)], \\ &\quad [P(x), \neg R(x)], [R(f(b))], [Q(x,y), \neg P(y)]\} \end{aligned}$$

Rezoluční pravidlo je pak definováno následovně: Mějme uspořádané klauzule

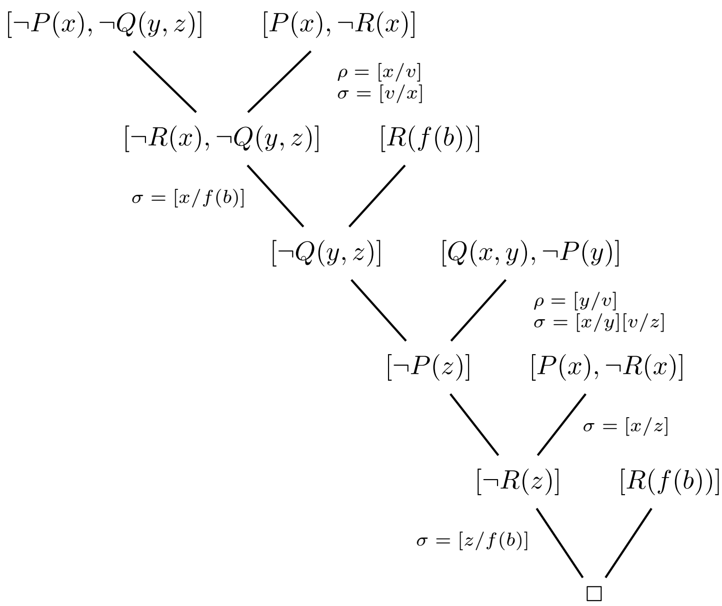
$$\begin{aligned} G &= [\neg A_1, \neg A_2, \dots, \neg A_n] \text{ a} \\ H &= [B_0, \neg B_1, \neg B_2, \dots, \neg B_m]. \end{aligned}$$

Rezolventou G a H pro $\phi = mgu(B_0, A_i)$ bude uspořádaná klauzule

$$[\neg A_1\phi, \neg A_2\phi, \dots, \neg A_{i-1}\phi, \neg B_1\phi, \neg B_2\phi, \dots, \neg B_m\phi, \neg A_{i+1}\phi, \dots, \neg A_n\phi].$$



e) SLD rezoluce (LD rezoluce se selekčním pravidlem) – jedná se o speciální případ LD rezoluce. K výběru literálu z předchozí rezolventy, podle kterého se bude rezolvovat dále, slouží tzv. výběrové pravidlo. Toto pravidlo je definováno tak, že volí vždy první literál z předchozí rezolventy.



$\square$

2 SLD-stromy a rezoluce v Prologu I

Příklad 2.1: Nalezněte rezoluční vyvrácení množiny klauzulí zadaných jako program a dotaz v Prologu.

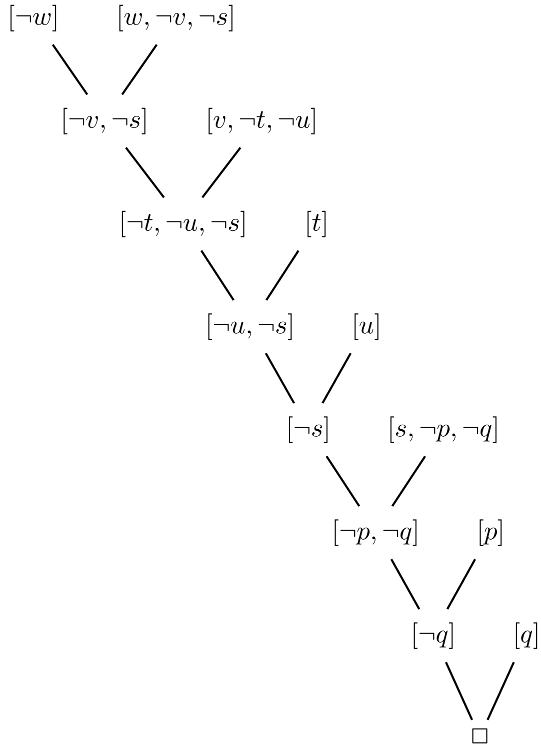
```
1. r :- p, q.        5. t.
2. s :- p, q.        6. q.
3. v :- t, u.        7. u.
4. w :- v, s.        8. p.

?- w.
```

Řešení 2.1: Program v jazyce Prolog nejprve převedeme na formuli výrokové logiky v konjunktivní normální formě. Přitom postupujeme tak, že každou programovou klauzuli tvaru `p :- q`$_1$`,...,q`$_n$`.` převedeme na uspořádanou klauzuli tvaru $[p, \neg q_1, \ldots, \neg q_n]$, fakt `q.` na klauzuli $[q]$ a dotaz `?- p`$_1$`,...,p`$_n$`.` na cílovou klauzuli $[\neg p_1, \ldots, \neg p_n]$. Získame tak formuli (resp. množinu klauzulí) v konjunktivní normální formě, která obsahuje pouze Hornovy klauzule. K vyvracení této množiny použijeme SLD-rezoluci.

$$\begin{aligned} S \;=\; & \{[r, \neg p, \neg q], [s, \neg p, \neg q], [v, \neg t, \neg u], \\ & [w, \neg v, \neg s], [t], [q], [u], [p], [\neg w]\} \end{aligned}$$

Aplikací popsaného postupu na program ze zadání obdržíme množinu klauzulí S a následující strom rezolučního vyvracení.



$\square$

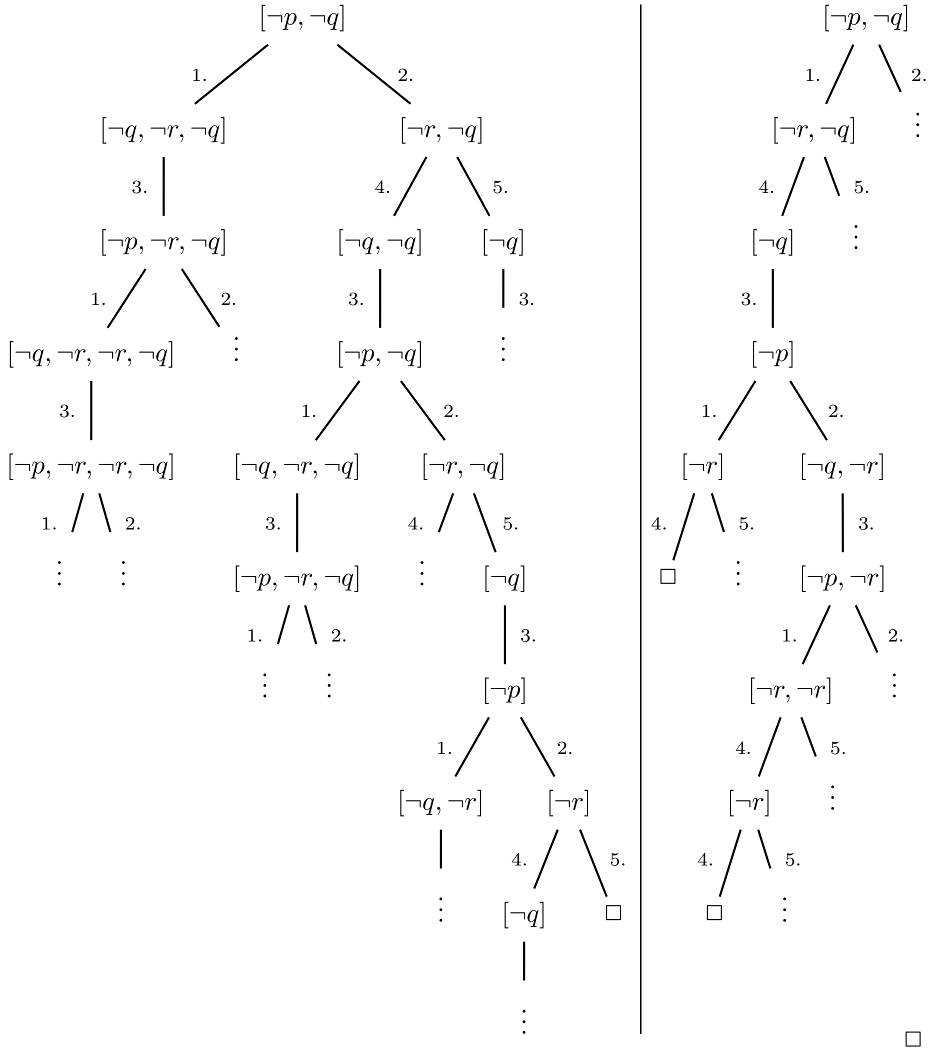
Příklad 2.2: Vytvořte SLD-strom pro následující program (`Program 1.`) a dotaz v Prologu a zjistěte, jak se projeví změna pořadí klauzulí (`Program 2.`) v definici programu na výsledné podobě SLD-stromu.

```
Program 1:                          Program 2:
 1. p :- q,r.      4. r :- q.        1. p :- r.        4. r.
 2. p :- r.        5. r.             2. p :- q,r.      5. r :- q.
 3. q :- p.                          3. q :- p.

 ?- p,q.
```

Řešení 2.2: SLD-strom konstruujeme tak, že do kořene dosadíme dotaz Q a pro každou možnou klauzuli C ze vstupní množiny, kterou lze použít k rezoluci, přidáme podstrom, který má v kořeni rezolventu Q a C (hranu vedoucí z Q k tomuto podstromu označíme číslem programové klauzule C).

Protože pracujeme s množinou Hornových klauzulí, obsahuje každý uzel stromu cílovou klauzuli (tj. klauzuli obsahující pouze negativní literály).



□

Příklad 2.3: Sestrojte SLD-stromy pro následující programy a dotazy v jazyce Prolog:

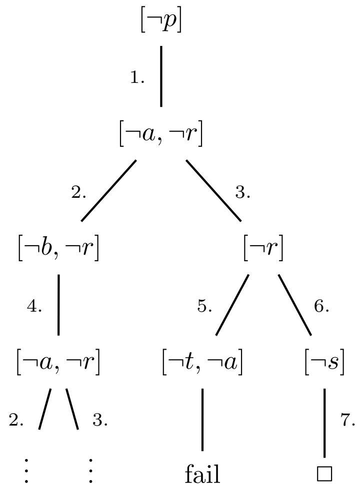
```
Program 1:                           Program 2:
 1. p :- a,r.  5. r :- t,a.           1. p :- s,t.  5.  r :- w.
 2. a :- b.    6. r :- s.             2. p :- q.    6.  r.
 3. a.         7. s.                  3. q.         7.  s.
 4. b :- a.                           4. q :- r.    8.  t :- w.

 ?- p.                                ?- p.
```

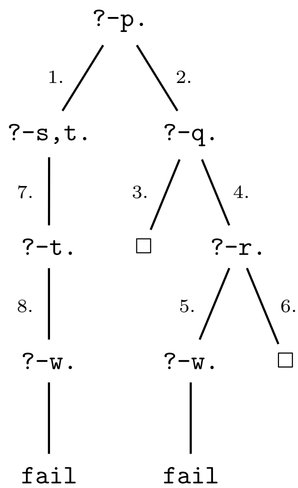
Řešení 2.3: Můžeme postupovat stejně jako při řešení předchozího příkladu. Programy převedeme na uspořádané klauzule, ze kterých poté konstruujeme samotný strom. Tento postup jsme použili pro `Program 1`.

Druhou možností je použít při konstrukci SLD-stromu notaci jazyka Prolog. Místo uspořádaných klauzulí tak budeme do jednotlivých uzlů zapisovat prologovské dotazy vzniklé aplikací rezolučního pravidla. Ušetříme si tak práci s přepisem pravidel na formule v CNF. Strom pak můžeme konstruovat velmi jednoduše tak, že první literál dotazu nahradíme pravou stranu programového pravidla, které používáme k rezoluci. Tuto metodu jsme použili pro konstrukci SLD-stromu pro `Program 2`.

`Program 1:`



`Program 2:`



□

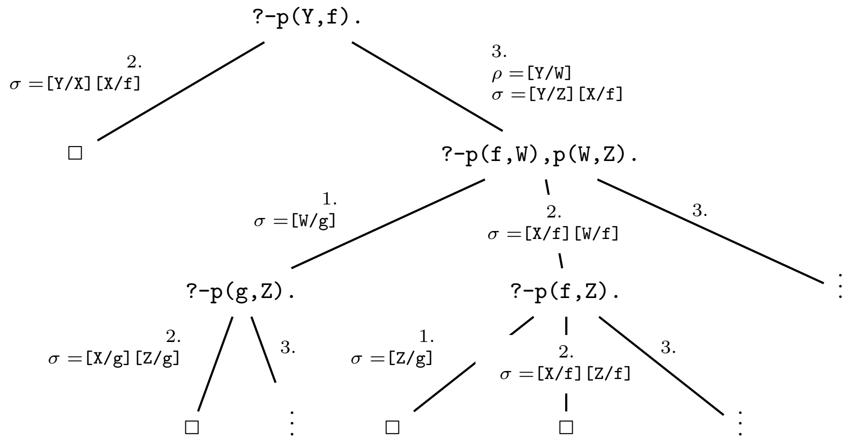
3 SLD-stromy a rezoluce v Prologu II

Příklad 3.1: Napište SLD-strom pro následující program a dotaz.

```
1. p(f,g).      3. p(Z,X) :- p(X,Y), p(Y,Z).
2. p(X,X).

?- p(Y,f).
```

Řešení 3.1: Pro řešení využijeme druhý způsob zápisu. Program využívá predikátovou logiku, musíme proto použít rezoluční pravidlo pro tuto logiku. Budeme tedy opět přejmenovávat proměnné (substituce ρ - aplikuje se na programovou klauzuli, ale stejně tak bychom mohli přejmenovávat i proměnné v cílové klauzuli) v případě, že se vyskytnou v obou klauzulích, a hledat nejobecnější unifikátor literálů (substituce σ), na kterých budeme rezoluci provádět.



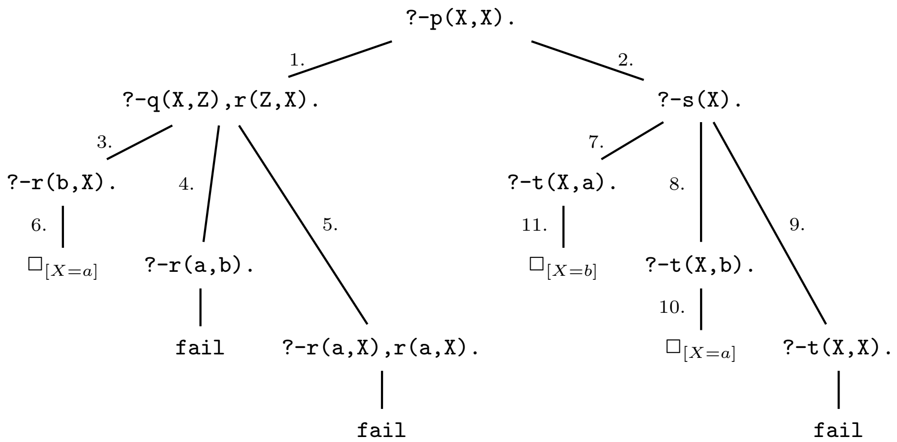
□

Příklad 3.2: Vytvořte SLD-strom všech možných řešení k následujícímu programu a dotazu:

```
1. p(X,Y) :- q(X,Z), r(Z,Y).
2. p(X,X) :- s(X).
3. q(X,b).
4. q(b,a).
5. q(X,a) :- r(a,X).
6. r(b,a).
7. s(X) :- t(X,a).
8. s(X) :- t(X,b).
9. s(X) :- t(X,X).
10. t(a,b).
11. t(b,a).

?- p(X,X).
```

Řešení 3.2: Sestrojíme následující SLD-strom v notaci jazyka Prolog.



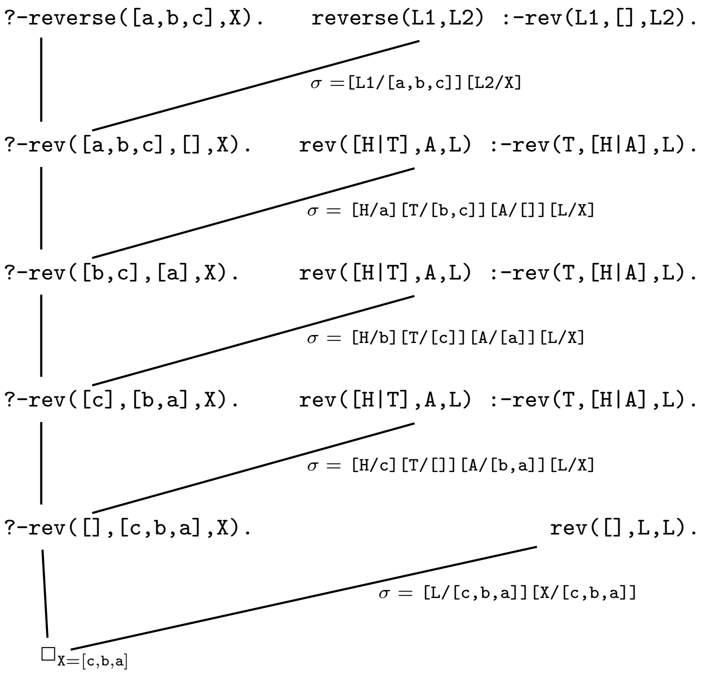
Ve stromu jsme pro přehlednost uvedli pouze čísla klauzulí z programu, které jsme použily při SLD-rezolučním vyvracení. Vynechali jsme tedy jak substituce použité pro přejmenování proměnných, tak nejobecnější unifikátory.

□

Příklad 3.3: Zkonstruujte SLD vyvrácení cíle `?- reverse([a,b,c],X).` za předpokladu, že máme nadefinován predikát `reverse/2` následovně:

```
reverse(L1,L2) :- rev(L1,[],L2).
rev([H|T],A,L) :- rev(T,[H|A],L).
rev([],L,L).
```

Řešení 3.3: Strom rezolučního vyvrácení dotazu `?- reverse([a,b,c],X).` vypadá následovně.



Zajímavé na tomto příkladu je zejména to, že jsme definovali program, který neřeší pouze rozhodovací problém (zda dotaz z programu vyplývá či nikoli), ale počítá novou hodnotu ze zadaných argumentů. Jedná se o program pro převrácení seznamu zadaného jako první argument.

Pokud na proměnnou `X` postupně aplikujeme všechny substituce v pořadí shora dolů (v tomto případě tedy pouze poslední substituci σ), získáváme substituci `X = [c,b,a]` uvedenou u prázdné klauzule. Tuto substituci nám Prolog také předloží jako odpověď na dotaz `?- reverse([a,b,c],X).`, tedy na dotaz, zda z uvedeného programu vyplývá formule $\exists$`X reverse([a,b,c],X)`. □