# Rezoluce a logické programování (pokračování)

## Uspořádané klauzule (*definite clauses*)

- Klauzule = množina literálů
- **Uspořádáná klauzule (*definite clause*) = posloupnost literálů**
  - nelze volně měnit pořadí literálů
- **Rezoluční princip pro uspořádané klauzule**:

$$\frac{\{\neg A_0, \ldots, \neg A_n\} \qquad \{B, \neg B_0, \ldots, \neg B_m\}}{\{\neg A_0, \ldots, \neg A_{i-1}, \neg B_0\rho, \ldots, \neg B_m\rho, \neg A_{i+1}, \ldots, \neg A_n\}\sigma}$$

  - **uspořádaná rezolventa**: $\{\neg A_0, \ldots, \neg A_{i-1}, \neg B_0\rho, \ldots, \neg B_m\rho, \neg A_{i+1}, \ldots, \neg A_n\}\sigma$
  - $\rho$ je přejmenování proměnných takové, že klauzule $\{A_0, \ldots, A_n\}$ a $\{B, B_0, \ldots, B_m\}\rho$ nemají společné proměnné
  - $\sigma$ je nejobecnější unifikátor pro $A_i$ a $B\rho$
  - **rezoluce je realizována na literálech** $\neg A_i\sigma$ **a** $B\rho\sigma$
  - je dodržováno pořadí literálů, tj. $\{\neg B_0\rho, \ldots, \neg B_m\rho\}\sigma$ **jde do uspořádané rezolventy přesně na pozici** $\neg A_i\sigma$



## Uspořádané klauzule II.

- Uspořádáné klauzule

$$\frac{\{\neg A_0, \ldots, \neg A_n\} \qquad \{B, \neg B_0, \ldots, \neg B_m\}}{\{\neg A_0, \ldots, \neg A_{i-1}, \neg B_0\rho, \ldots, \neg B_m\rho, \neg A_{i+1}, \ldots, \neg A_n\}\sigma}$$

  Hornovy klauzule

$$\frac{:-A_0, \ldots, A_n. \qquad B :- B_0, \ldots, B_m.}{:-(A_0, \ldots, A_{i-1}, B_0\rho, \ldots, B_m\rho, A_{i+1}, \ldots, A_n)\sigma.}$$

- Příklad:

$$\frac{\{\neg s(X), \neg t(1), \neg u(X)\} \qquad \{t(Z), \neg q(Z,X), \neg r(3)\}}{\{\neg s(X), \neg q(1,A), \neg r(3), \neg u(X)\}}$$

$$\frac{:-s(X), t(1), u(X). \qquad t(Z) :- q(Z,X), r(3).}{:-s(X), q(1,A), r(3), u(X).}$$

$$\rho = [X/A] \qquad \sigma = [Z/1]$$



## LD-rezoluce

- **LD-rezoluční vyvrácení** množiny uspořádaných klauzulí $P \cup \{G\}$ je posloupnost $\langle G_0, C_0\rangle, \ldots, \langle G_n, C_n\rangle$ taková, že
  - $G_i, C_i$ jsou uspořádané klauzule
  - $G = G_0$
  - $G_{n+1} = \square$
  - $G_i$ je uspořádaná cílová klauzule
  - $C_i$ je přejmenování klauzule z $P$
    - $C_i$ neobsahuje proměnné, které jsou v $G_j, j \leq i$ nebo v $C_k, k \leq i$
  - $G_{i+1}, 0 \leq i \leq n$ je uspořádaná rezolventa $G_i$ a $C_i$
- LD-rezoluce: korektní a úplná

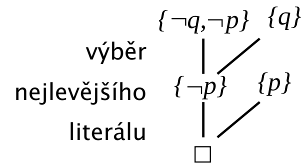
# SLD-rezoluce

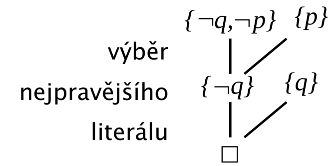
- **Lineární rezoluce se selekčním pravidlem = SLD-rezoluce (*Selected Linear resolution for Definite clauses*)**
  - rezoluce
  - **Selekční** pravidlo
  - **Lineární** rezoluce
  - **Definite** (uspořádané) klauzule
  - vstupní rezoluce
- **Selekční pravidlo** $R$ je funkce, která každé neprázdné klauzuli $C$ přiřazuje nějaký z jejích literálů $R(C) \in C$
  - při rezoluci vybírám z klauzule literál určený selekčním pravidlem
- Pokud se $R$ neuvádí, pak se předpokládá výběr **nejlevějšího literálu**
  - nejlevější literál vybírá i Prolog



# Lineární rezoluce se selekčním pravidlem

- $P = \{\{p\}, \{p, \neg q\}, \{q\}\}, \quad G = \{\neg q, \neg p\}$



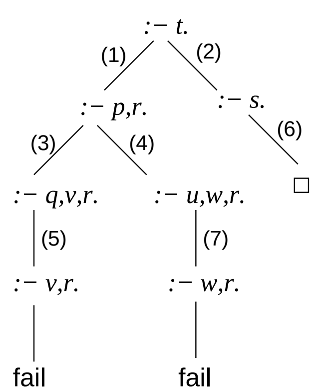

- **SLD-rezoluční vyvrácení** $P \cup \{G\}$ pomocí selekčního pravidla $R$ je LD-rezoluční vyvrácení $\langle G_0, C_0 \rangle, \ldots, \langle G_n, C_n \rangle$ takové, že $G = G_0, G_{n+1} = \square$ a $R(G_i)$ je literál rezolvovaný v kroku $i$
- SLD-rezoluce – korektní, úplná
- Efektivita SLD-rezoluce je závislá na
  - selekčním pravidle $R$
  - způsobu výběru příslušné programové klauzule pro tvorbu rezolventy
    - v Prologu se vybírá vždy klauzule, která je v programu první



# Příklad: SLD-strom

| | |
|---|---|
| $t : -p, r.$ | (1) |
| $t : -s.$ | (2) |
| $p : -q, v.$ | (3) |
| $p : -u, w.$ | (4) |
| $q.$ | (5) |
| $s.$ | (6) |
| $u.$ | (7) |

$: -t.$



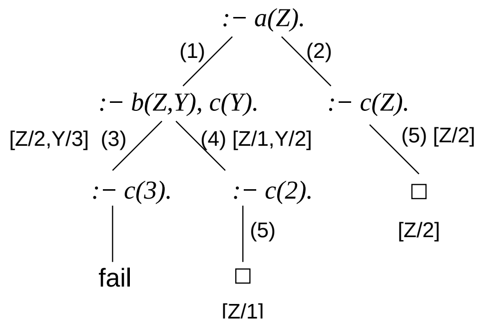



# Strom výpočtu (SLD-strom)

- **SLD-strom** je strom tvořený všemi možnými výpočetními posloupnostmi logického programu $P$ vzhledem k cíli $G$
- kořeny stromy jsou programové klauzule a cílová klauzule $G$
- v uzlech jsou rezolventy
- výchozím kořenem rezoluce je cílová klauzule $G$
- listy jsou dvojího druhu:
  - označené prázdnou klauzulí – jedná se o **úspěšné uzly** (*succes nodes*)
  - označené neprázdnou klauzulí – jedná se o **neúspěšné uzly** (*failure nodes*)
- úplnost SLD-rezoluce zaručuje **existenci** cesty od kořene k úspěšnému uzlu pro každý možný výsledek příslušející cíli $G$

# Příklad: SLD-strom a výsledná substituce

$:-a(Z).$




$a(X):-b(X,Y),c(Y).$ (1)
$a(X):-c(X).$ (2)
$b(2,3).$ (3)
$b(1,2).$ (4)
$c(2).$ (5)

Cvičení:

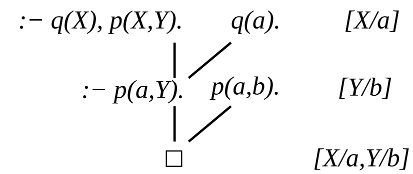
$p(B):-q(A,B),r(B).$ ve výsledné substituci jsou pouze proměnné z dotazu
$p(A):-q(A,A).$ výsledné substituce jsou $[Z/1]$ a $[Z/2]$
$q(a,a).$ nezajímá mě substituce $[Y/2]$
$q(a,b).$
$r(b).$



# Výsledná substituce (*answer substitution*)

$q(a).$
$p(a,b).$

$:-q(X), p(X,Y).$
$X=a,\ Y=b$




- Každý krok SLD-rezoluce vytváří novou unifikační substituci $\theta_i$
  ⇒ potenciální instanciace proměnné ve vstupní cílové klauzuli
- **Výsledná substituce** (*answer substitution*)

  $\theta = \theta_0\theta_1 \cdots \theta_n$ složení unifikací



# Význam SLD-rezolučního vyvrácení $P \cup \{G\}$

- Množina $P$ programových klauzulí, cílová klauzule $G$
- **Dokazujeme nesplnitelnost**

  (1) $P \wedge (\forall \vec{X})(\neg G_1(\vec{X}) \vee \neg G_2(\vec{X}) \vee \cdots \vee \neg G_n(\vec{X}))$
  kde $G = \{\neg G_1, \neg G_2, \cdots, \neg G_n\}$ a $\vec{X}$ je vektor proměnných v $G$
  nesplnitelnost (1) je ekvivalentní tvrzení (2) a (3)

  (2) $P \vdash \neg G$

  (3) $P \vdash (\exists \vec{X})(G_1(\vec{X}) \wedge \cdots \wedge G_n(\vec{X}))$

  a jedná se tak o **důkaz existence vhodných objektů**, které na základě vlastností množiny $P$ splňují konjunkci literálů v cílové klauzuli
- Důkaz nesplnitelnosti $P \cup \{G\}$ znamená **nalezení protipříkladu**
  ten pomocí SLD-stromu **konstruuje termy (odpověď)** splňující konjunkci v (3)



# Výpočetní strategie

- **Korektní výpočetní strategie** prohledávání stromu výpočtu musí zaručit, že se každý (konečný) výsledek nalézt v konečném čase
- Korektní výpočetní strategie = **prohledávání stromu do šířky**
  - exponenciální paměťová náročnost
  - složité řídící struktury
- Použitelná výpočetní strategie = **prohledávání stromu do hloubky**
  - jednoduché řídící struktury (zásobník)
  - lineární paměťová náročnost
  - **není ale úplná**: nenalezne vyvrácení i když existuje
    - procházení nekonečné větve stromu výpočtu
      ⇒ na nekonečných stromech dojde k zacyklení
    - nedostaneme se tak na jiné existující úspěšné uzly

# SLD-rezoluce v Prologu: úplnost

- **Prolog**: prohledávání stromu do hloubky
  
  ⇒ **neúplnost** použité výpočetní strategie

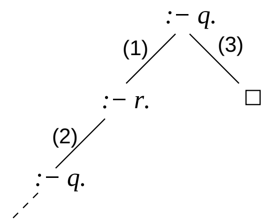
- Implementace SLD-rezoluce v Prologu
  - **není úplná**

  logický program: $q : -r.$ (1)

  $r : -q.$ (2)

  $q.$ (3)

  dotaz: $: -q.$






# Test výskytu

- Kontrola, zda se proměnná vyskytuje v termu, kterým ji substituujeme
  - dotaz $: -a(B, B).$
  - logický program: $a(X, f(X)).$
  - vede k: $[B/X]$, $[X/f(X)]$
- Unifikátor pro $g(X_1, \dots, X_n)$ a $g(f(X_0, X_0), f(X_1, X_1), \dots, f(X_{n-1}, X_{n-1}))$

  $X_1 = f(X_0, X_0), \quad X_2 = f(X_1, X_1), \dots, \quad X_n = f(X_{n-1}, X_{n-1})$

  $X_2 = f(f(X_0, X_0), f(X_0, X_0)), \dots$

  délka termu pro $X_k$ exponenciálně narůstá

  $\Longrightarrow$ **exponenciální složitost** na ověření kontroly výskytu
- Test výskytu se **při unifikaci v Prologu neprovádí**
- Důsledek: $? - X = f(X)$ uspěje s $X = f(f(f(f(f(f(f(f(f(f(...))))))))))$



# SLD-rezoluce v Prologu: korektnost

- Implementace SLD-rezoluce v Prologu nepoužívá při unifikaci test výskytu

  $\Longrightarrow$ **není korektní**

(1) $t(X) : -p(X, X).$ $: -t(X).$

$p(X, f(X)).$ $X = f(f(f(f(...)))))))))$ problém se projeví

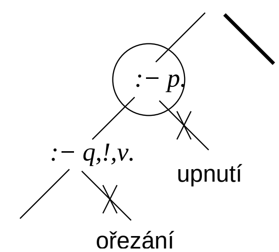
(2) $t : -p(X, X).$ $: -t.$

$p(X, f(X)).$ yes dokazovací systém nehledá unifikátor pro $X$ a $f(X)$

- Řešení: problém typu (2) převést na problém typu (1) ?
  - každá proměnná v hlavě klauzule se objeví i v těle, aby se vynutilo hledání unifikátoru (přidáme $X = X$ pro každou $X$, která se vyskytuje pouze v hlavě)
  
    $t : -p(X, X).$
    
    $p(X, f(X)) : -X = X.$
  - optimalizace v kompilátoru mohou způsobit opět odpověď' „yes"



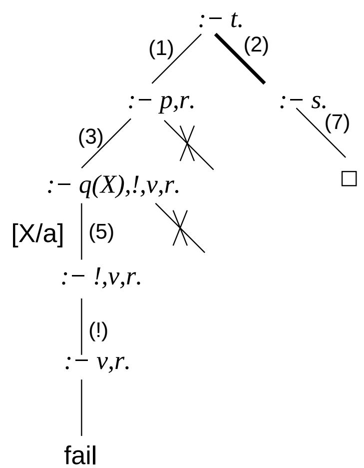
# Řízení implementace: řez




- nemá ale žádnou deklarativní sémantiku
- místo toho **mění implementaci programu**
- $p : -q, !, v.$
- snažíme se splnit $q$
- řez se syntakticky chová jako kterýkoliv jiný literál
- pokud uspěji
  
  ⇒ přeskočím řez a pokračuji jako by tam řez nebyl
- pokud ale **neuspěji (a tedy i při backtrackingu) a vracím se přes řez**
  
  ⇒ **vracím se až na rodiče** $: -p.$ a zkouším další větev
  
  ⇒ nezkouším tedy další možnosti, jak splnit $p$ upnutí
  
  ⇒ a nezkouším ani další možnosti, jak splnit $q$ v SLD-stromu ořezání

## Příklad: řez

$t:-p,r.$ (1)
$t:-s.$ (2)
$p:-q(X),!,v.$ (3)
$p:-u,w.$ (4)
$q(a).$ (5)
$q(b).$ (6)
$s.$ (7)
$u.$ (8)



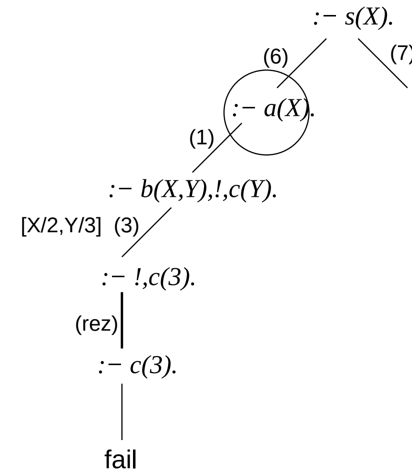



## Příklad: řez II

$a(X):-b(X,Y),!,c(Y).$ (1)
$a(X):-c(X).$ (2)
$b(2,3).$ (3)
$b(1,2).$ (4)
$c(2).$ (5)

$s(X):-a(X).$ (6)
$s(X):-p(X).$ (7)

$p(B):-q(A,B),r(B).$ (8)
$p(A):-q(A,A).$ (9)
$q(a,a).$ (10)
$q(a,b).$ (11)
$r(b).$ (12)



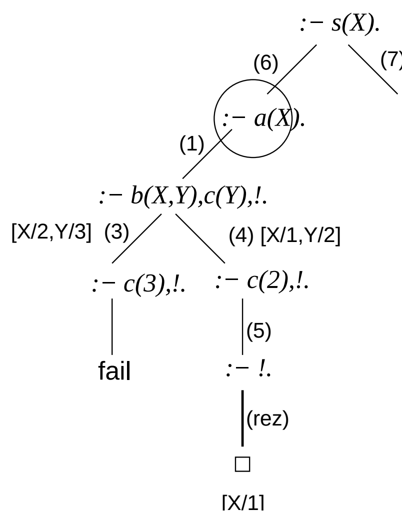



## Příklad: řez III

$a(X):-b(X,Y),c(Y),!.$ (1)
$a(X):-c(X).$ (2)
$b(2,3).$ (3)
$b(1,2).$ (4)
$c(2).$ (5)

$s(X):-a(X).$ (6)
$s(X):-p(X).$ (7)

$p(B):-q(A,B),r(B).$ (8)
$p(A):-q(A,A).$ (9)
$q(a,a).$ (10)
$q(a,b).$ (11)
$r(b).$ (12)






## Operační a deklarativní semantika

# Operační sémantika

- **Operační sémantikou** logického programu $P$ rozumíme množinu $O(P)$ všech atomických formulí bez proměnných, které lze pro nějaký cíl $G^1$ odvodit nějakým rezolučním důkazem ze vstupní množiny $P \cup \{G\}$.

  [1] tímto výrazem jsou míněny všechny cíle, pro něž zmíněný rezoluční důkaz existuje.
- **Deklarativní sémantika** logického programu $P$ ???



# Opakování: interpretace

- **Interpretace** $\mathcal{I}$ jazyka $\mathcal{L}$ je dána univerzem $\mathcal{D}$ a zobrazením, které přiřadí konstantě $c$ prvek $\mathcal{D}$, funkčnímu symbolu $f/n$ $n$-ární operaci v $\mathcal{D}$ a predikátovému symbolu $p/n$ $n$-ární relaci.
  - příklad: $F = \{\{f(a,b) = f(b,a)\}, \{f(f(a,a),b) = a\}\}$
    interpretace $\mathcal{I}_1$: $\mathcal{D} = \mathbb{Z}, a := 1, b := -1, f := " + "$
- Interpretace se nazývá **modelem** formule, je-li v ní tato formule pravdivá
  - interpretace množiny $\mathbb{N}$ s obvyklými operacemi je modelem formule ( 0 + s(0) = s(0) )



# Herbrandovy interpretace

- Omezení na obor skládající se ze **symbolických výrazů tvořených z predikátových a funkčních symbolů daného jazyka**
  - při zkoumání pravdivosti není nutné uvažovat modely nad všemi interpretacemi
- **Herbrandovo univerzum**: množina všech termů bez proměnných, které mohou být tvořeny funkčními symboly a konstantami daného jazyka
- **Herbrandova interpretace**: libovolná interpretace, která přiřazuje
  - proměnným prvky Herbrandova univerza
  - konstantám sebe samé
  - funkčním symbolům funkce, které symbolu f pro argumenty $t_1, \cdots, t_n$ přiřadí term f$(t_1, \cdots, t_n)$
  - predikátovým symbolům libovolnou funkci z Herbrand. univerza do pravdivostních hodnot
- **Herbrandův model** množiny uzavřených formulí $\mathcal{P}$:
  Herbrandova interpretace taková, že každá formule z $\mathcal{P}$ je v ní pravdivá.



# Specifikace Herbrandova modelu

- Herbrandovy interpretace mají předdefinovaný význam funktorů a konstant
- Pro specifikaci Herbrandovy interpretace tedy stačí zadat relace pro každý predikátový symbol
- Příklad: Herbrandova interpretace a Herbrandův model množiny formulí

```
lichy(s(0)).                    % (1)
lichy(s(s(X))) :- lichy(X). % (2)
```

  - $\mathcal{I}_1 = \emptyset$ — není model (1)
  - $\mathcal{I}_2 = \{lichy(s(0))\}$ — není model (2)
  - $\mathcal{I}_3 = \{lichy(s(0)), lichy(s(s(s(0))))\}$ — není model (2)
  - $\mathcal{I}_4 = \{lichy(s^n(0)) | n \in \{1, 3, 5, 7, \ldots\}\}$ — Herbrandův model (1) i (2)
  - $\mathcal{I}_5 = \{lichy(s^n(0)) | n \in \mathbb{N}\}$ — Herbrandův model (1) i (2)

## Příklad: Herbrandovy interpretace

```
rodic(a,b).
rodic(b,c).
predek(X,Y) :- rodic(X,Y).
predek(X,Z) :- rodic(X,Y), predek(Y,Z).
```

$\mathcal{I}_1 = \{rodic(a,b), rodic(b,c), predek(a,b), predek(b,c), predek(a,c)\}$

$\mathcal{I}_2 = \{rodic(a,b), rodic(b,c),$
$\quad predek(a,b), predek(b,c), predek(a,c), predek(a,a)\}$

$\mathcal{I}_1$ i $\mathcal{I}_2$ jsou Herbrandovy modely klauzulí



## Deklarativní a operační sémantika

- Je-li $S$ množina programových klauzulí a $M$ libovolná množina Herbrandových modelů $S$, pak **průnik těchto modelů** je opět Herbrandův model množiny $S$.
- **Důsledek:**
  Existuje **nejmenší Herbrandův model** množiny $S$, který značíme $M(S)$.
- **Deklarativní sémantikou** logického programu $P$ rozumíme jeho minimální Herbrandův model $M(P)$.
- **Operační sémantikou** logického programu $P$ rozumíme množinu $O(P)$ všech atomických formulí bez proměnných, které lze pro nějaký cíl $\mathcal{G}$[1] odvodit nějakým rezolučním důkazem ze vstupní množiny $P \cup \{\mathcal{G}\}$.

  [1]tímto výrazem jsou míněny všechny cíle, pro něž zmíněný rezoluční důkaz existuje.
- Pro libovolný logický program $P$ platí $\quad M(P) = O(P)$