

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 $P \cup \{G\}$.

¹tímto výrazem jsou míněny všechny cíle, pro něž zmíněný rezoluční důkaz existuje.

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 $P \cup \{G\}$.

¹tímto 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 $\mathcal{I}_1: \mathcal{D} = \mathbb{Z}, a := 1, b := -1, f := "+"$

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

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, \dots, t_n přiřadí term $f(t_1, \dots, t_n)$
 - predikátovým symbolům libovolnou funkci z Herbrand. univerza do pravdivostních hodnot

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, \dots, t_n přiřadí term $f(t_1, \dots, t_n)$
 - 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

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

`Tichy(s(0)). % (1)`

`Tichy(s(s(X))) :- Tichy(X). % (2)`

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

`Tichy(s(0)). % (1)`

`Tichy(s(s(X))) :- Tichy(X). % (2)`

● $\mathcal{I}_1 = \emptyset$

není model (1)

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

`Tichy(s(0)).` % (1)

`Tichy(s(s(X))) :- Tichy(X).` % (2)

● $\mathcal{I}_1 = \emptyset$ není model (1)

● $\mathcal{I}_2 = \{lichy(s(0))\}$ není model (2)

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

`Tichy(s(0)). % (1)`

`Tichy(s(s(X))) :- Tichy(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)

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

`Tichy(s(0)).` % (1)

`Tichy(s(s(X))) :- Tichy(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)
- $\mathcal{I}_4 = \{lichy(s^n(0)) \mid n \in \{1, 3, 5, 7, \dots\}\}$ Herbrandův model (1) i (2)

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

$\text{Tichy}(s(0)).$ % (1)

$\text{Tichy}(s(s(X))) \text{ :- } \text{Tichy}(X).$ % (2)

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

Příklad: Herbrandovy interpretace

`rodic(a,b).`

`rodic(b,c).`

`predek(X,Y) :- rodic(X,Y).`

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

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

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

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

Cvičení: Napište minimální Herbrandův model pro následující logický program.

`muz(petr). muz(pavel). zena(o1ga). zena(jitka).`

`pary(X,Y) :- zena(X), muz(Y).`

Uveďte další model tohoto programu, který není minimální.

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 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 $M(P)$.

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 $M(P)$.
- Připomenutí: **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 $P \cup \{G\}$.
¹tímto výrazem jsou míněny všechny cíle, pro něž zmíněný rezoluční důkaz existuje.

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 $M(P)$.
- Připomenutí: **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 $P \cup \{G\}$.
¹tímto výrazem jsou míněny všechny cíle, pro něž zmíněný rezoluční důkaz existuje.
- Pro libovolný logický program P platí $M(P) = O(P)$

SLD rezoluce v Prologu (pokračování)

Výpočetní strategie

- **Korektní výpočetní strategie** prohledávání stromu výpočtu musí zaručit, že se každý (konečný) výsledek nalézt v konečném čase

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 řídicí struktury

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 řídicí struktury
- Použitelná výpočetní strategie = **prohledávání stromu do hloubky**
 - jednoduché řídicí 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

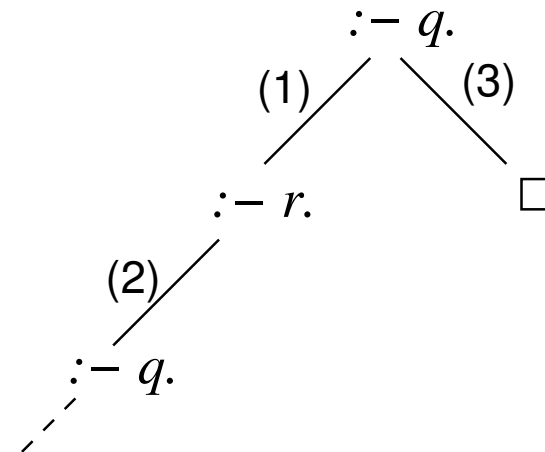
- **není úplná**

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: $a(X, f(X))$.

● by vedl 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

Test výskytu

- Kontrola, zda se proměnná vyskytuje v termu, kterým ji substituujeme

- dotaz : $-a(B, B)$.

- logický program: $a(X, f(X))$.

- by vedl 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

⇒ **exponenciální složitost** na ověření kontroly výskytu

- Test výskytu se **při unifikaci v Prologu neprovádí**

- Důsledek: ? – $X = f(X)$ uspěje s $X = f(f(f(f(f(f(f(f(f(...))))))))))$

SLD-rezoluce v Prologu: korektnost

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

⇒ **není korektní**

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

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

SLD-rezoluce v Prologu: korektnost

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

⇒ **není korektní**

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

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

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

$p(X, f(X)). \quad \text{yes}$ dokazovací systém nehledá unifikátor pro X a $f(X)$

SLD-rezoluce v Prologu: korektnost

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

⇒ **není korektní**

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

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

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

$p(X, f(X)). \quad \text{yes}$ dokazovací systém nehledá unifikátor pro X a $f(X)$

- Řešení: problém typu (2) převést na problém typu (1) ?

SLD-rezoluce v Prologu: korektnost

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

⇒ **není korektní**

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

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

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

$p(X, f(X)). \quad \text{yes}$ 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)

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

$p(X, f(X)) : -X = X.$

SLD-rezoluce v Prologu: korektnost

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

⇒ **není korektní**

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

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

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

$p(X, f(X)). \quad \text{yes}$ 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)

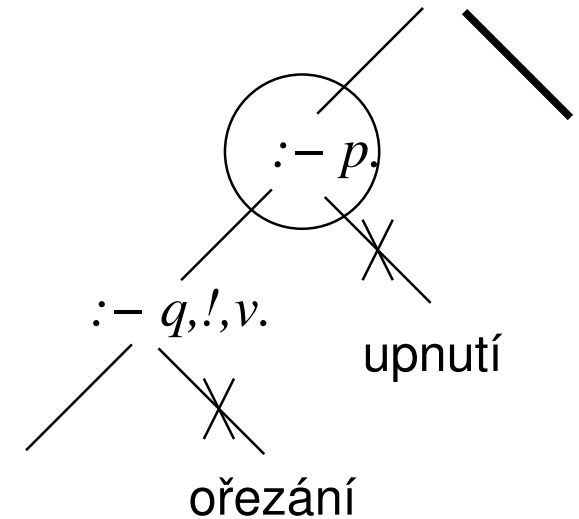
$t : -p(X, X).$

$p(X, f(X)) : -X = X.$

- optimalizace v kompilátoru mohou způsobit opět odpověď „yes”

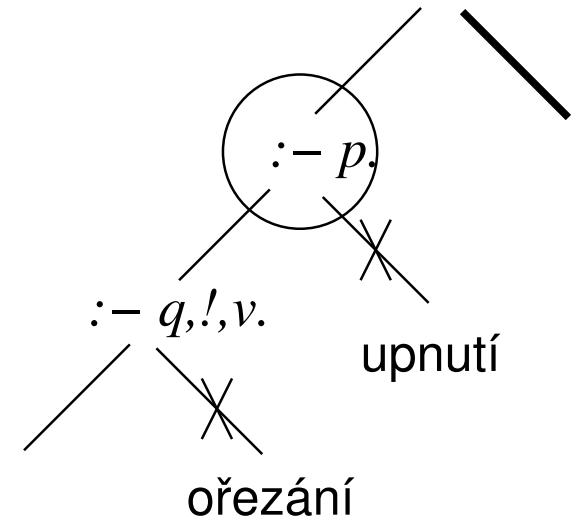
Řízení implementace: řez

- řez se syntakticky chová jako kterýkoliv jiný literál
- nemá ale žádnou deklarativní sémantiku
- místo toho **mění implementaci programu**
- $p : -q, !, v.$



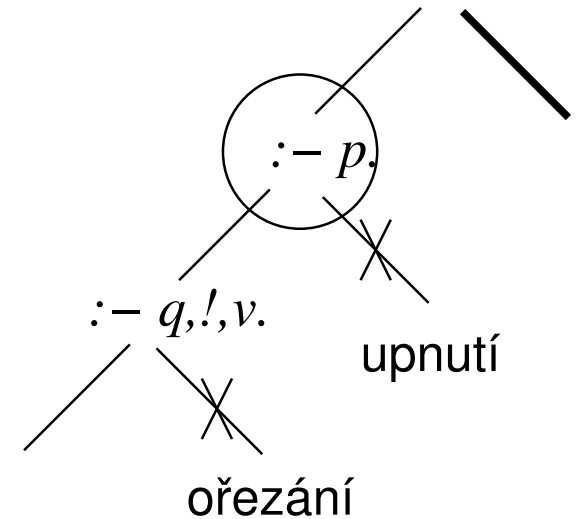
Řízení implementace: řez

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



Řízení implementace: řez

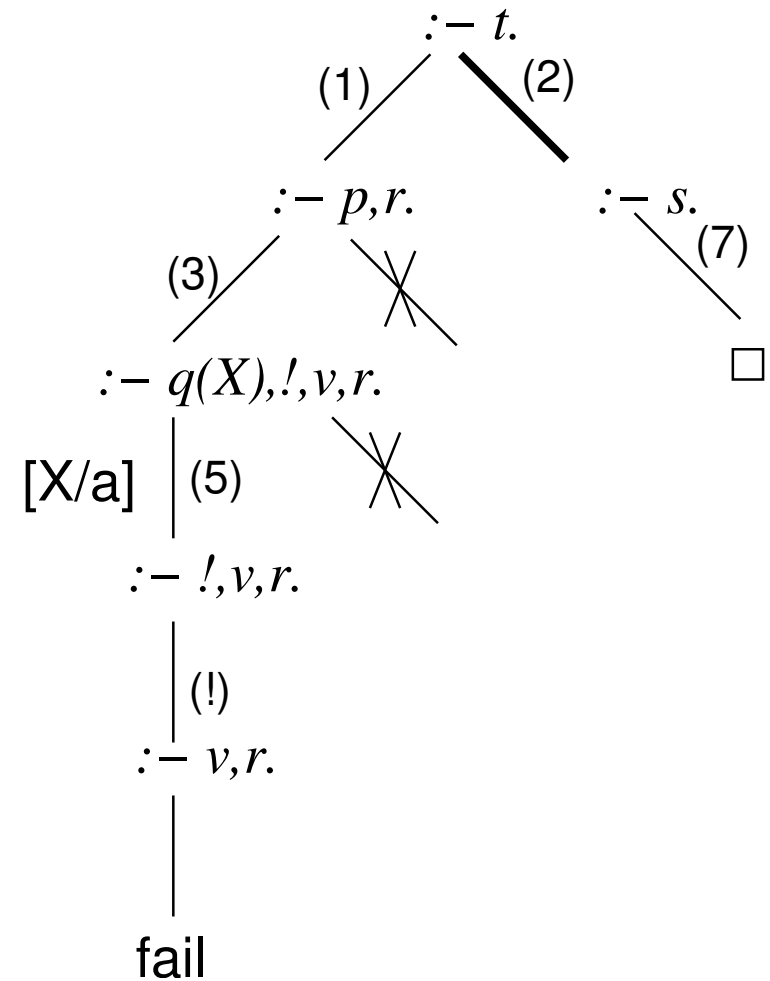
- řez se syntakticky chová jako kterýkoliv jiný literál
- nemá ale žádnou deklarativní sémantiku
- místo toho **mění implementaci programu**
- $p : -q, !, \nu.$
- snažíme se splnit q
- pokud uspějí
 - ⇒ přeskočím řez a pokračuji jako by tam řez nebyl
- pokud ale **neuspějí (a tedy i při backtrackingu)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



upnutí
ořezání

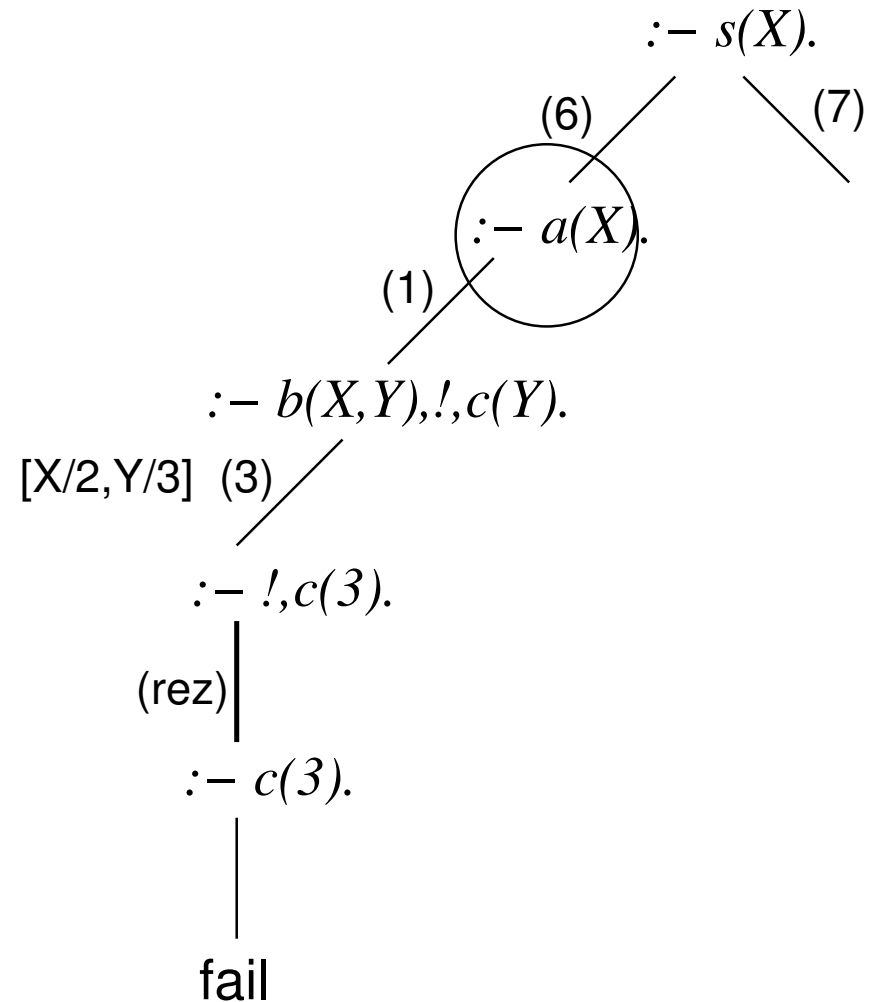
Příklad: řez

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



Příklad: řez II

- $a(X) : \neg b(X, Y), !, c(Y).$ (1)
 $a(X) : \neg c(X).$ (2)
 $b(2, 3).$ (3)
 $b(1, 2).$ (4)
 $c(2).$ (5)
- $s(X) : \neg a(X).$ (6)
 $s(X) : \neg p(X).$ (7)
- $p(B) : \neg q(A, B), r(B).$ (8)
 $p(A) : \neg q(A, A).$ (9)
 $q(a, a).$ (10)
 $q(a, b).$ (11)
 $r(b).$ (12)

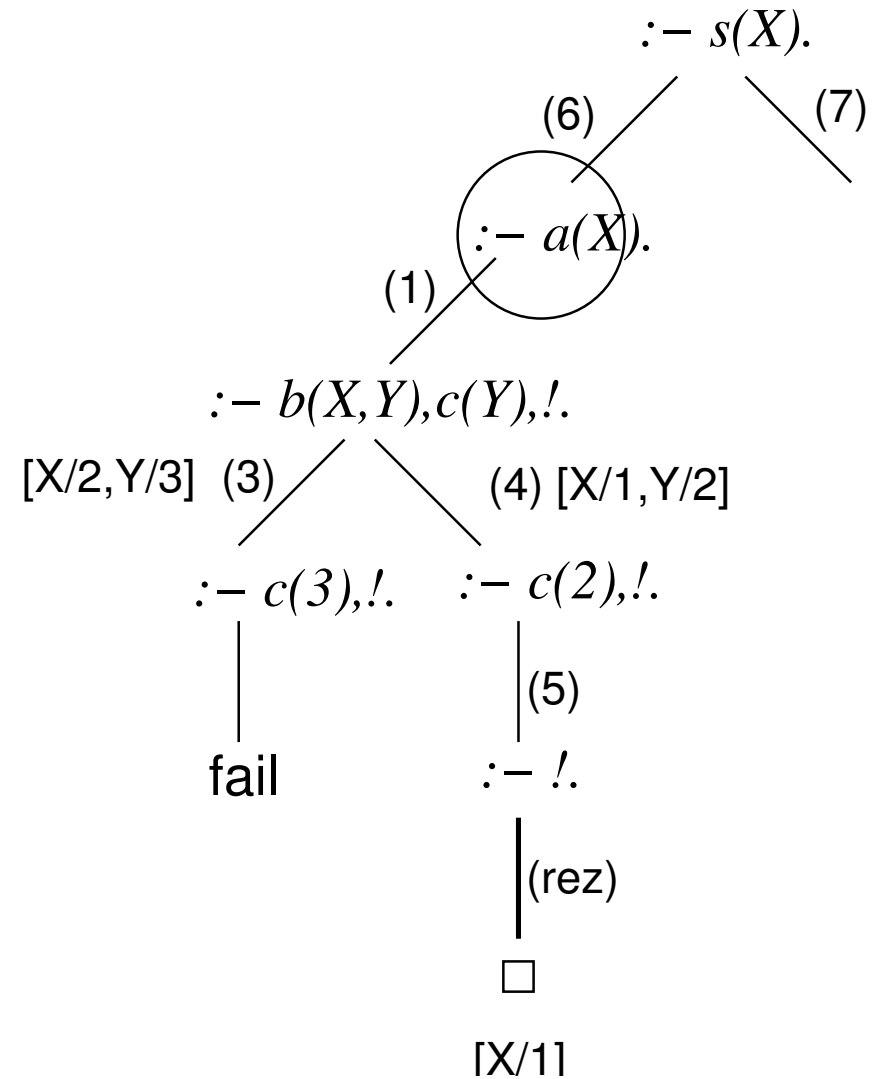


Příklad: řez III

- $a(X) : -b(X, Y), c(Y), !.$ (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)



Negace v logickém programování

Negativní znalost

- logické programy vyjadřují **pozitivní znalost**
- **negativní literály**: pozice určena definicí Hornových klauzulí
 - ⇒ nelze vyvodit **negativní** informaci z logického programu
- každý predikát definuje úplnou relaci
- negativní literál **není** logickým důsledkem programu

Negativní znalost

- logické programy vyjadřují **pozitivní znalost**
- **negativní literály**: pozice určena definicí Hornových klauzulí
 - ⇒ nelze vyvodit **negativní** informaci z logického programu
 - každý predikát definuje úplnou relaci
 - negativní literál **není** logickým důsledkem programu
- relace vyjádřeny explicitně v nejmenším Herbrandově modelu
 - $nad(X, Y) : -na(X, Y). \quad na(c, b).$
 - $nad(X, Y) : -na(X, Z), nad(Z, Y). \quad na(b, a).$
 - nejmenší Herbrandův model: $\{na(b, a), na(c, b), nad(b, a), nad(c, b), nad(c, a)\}$

Negativní znalost

- logické programy vyjadřují **pozitivní znalost**
- **negativní literály**: pozice určena definicí Hornových klauzulí
 - ⇒ nelze vyvodit **negativní** informaci z logického programu
 - každý predikát definuje úplnou relaci
 - negativní literál **není** logickým důsledkem programu
- relace vyjádřeny explicitně v nejmenším Herbrandově modelu
 - $nad(X, Y) : \neg na(X, Y). \quad na(c, b).$
 - $nad(X, Y) : \neg na(X, Z), nad(Z, Y). \quad na(b, a).$
 - nejmenší Herbrandův model: $\{na(b, a), na(c, b), nad(b, a), nad(c, b), nad(c, a)\}$
- ani program ani model nezahrnují negativní informaci
 - a není nad c , a není na c
 - i v realitě je negativní informace vyjádřena explicitně zřídka, např. jízdní řád

Předpoklad uzavřeného světa

- neexistence informace chápána jako opak:
předpoklad uzavřeného světa (*closed world assumption, CWA*)
- převzato z databází
- určitý vztah platí **pouze** když je vyvoditelný z programu.
- „odvozovací pravidlo” (A je (uzavřený) term): $\frac{P \neq A}{\neg A}$ (CWA)

Předpoklad uzavřeného světa

- neexistence informace chápána jako opak:
předpoklad uzavřeného světa (*closed world assumption, CWA*)
- převzato z databází
- určitý vztah platí **pouze** když je vyvoditelný z programu.
- „odvozovací pravidlo” (A je (uzavřený) term): $\frac{P \not\equiv A}{\neg A}$ (CWA)
- pro SLD-rezoluci: $P \not\equiv nad(a, c)$, tedy lze podle CWA odvodit $\neg nad(a, c)$

Předpoklad uzavřeného světa

- neexistence informace chápána jako opak:
předpoklad uzavřeného světa (*closed world assumption, CWA*)
- převzato z databází
- určitý vztah platí **pouze** když je vyvoditelný z programu.
- „odvozovací pravidlo” (A je (uzavřený) term): $\frac{P \neq A}{\neg A}$ (CWA)
- pro SLD-rezoluci: $P \neq nad(a, c)$, tedy lze podle CWA odvodit $\neg nad(a, c)$
- problém: není rozhodnutelné, zda daná atomická formule je logickým důsledkem daného logického programu.
 - nelze tedy určit, zda pravidlo CWA je aplikovatelné nebo ne
- CWA v logickém programování obecně nepoužitelná.

Negace jako neúspěch (*negation as failure*)

- slabší verze CWA: **definitivně neúspěšný (*finitely failed*) SLD-strom** cíle : $\neg A$
: $\neg A$ má definitivně (konečně) neúspěšný SLD-strom **(*negation as failure*, NF)**
 $\neg A$
- **normální cíl**: cíl obsahující i negativní literály
 - : $\neg nad(c, a)$, $\neg nad(b, c)$.

Negace jako neúspěch (*negation as failure*)

- slabší verze CWA: **definitivně neúspěšný (*finitely failed*) SLD-strom** cíle : $\neg A$
: $\neg A$ má definitivně (konečně) neúspěšný SLD-strom **(*negation as failure*, NF)**
 $\neg A$
- **normální cíl**: cíl obsahující i negativní literály
 - : $\neg nad(c, a)$, $\neg nad(b, c)$.
- rozdíl mezi CWA a NF
 - program $nad(X, Y) : \neg nad(X, Y)$, cíl : $\neg \neg nad(b, c)$
 - neexistuje odvození cíle podle NF, protože SLD-strom : $\neg nad(b, c)$ je nekonečný
 - existuje odvození cíle podle CWA, protože neexistuje vyvrácení : $\neg nad(b, c)$

Negace jako neúspěch (*negation as failure*)

- slabší verze CWA: **definitivně neúspěšný (*finitely failed*) SLD-strom** cíle : $\neg A$
: $\neg A$ má definitivně (konečně) neúspěšný SLD-strom **(*negation as failure*, NF)**
 $\neg A$
- **normální cíl**: cíl obsahující i negativní literály
 - : $\neg nad(c, a)$, $\neg nad(b, c)$.
- rozdíl mezi CWA a NF
 - program $nad(X, Y)$: $\neg nad(X, Y)$, cíl : $\neg \neg nad(b, c)$
 - neexistuje odvození cíle podle NF, protože SLD-strom : $\neg nad(b, c)$ je nekonečný
 - existuje odvození cíle podle CWA, protože neexistuje vyvrácení : $\neg nad(b, c)$
- CWA i NF jsou nekorektní: A není logickým důsledkem programu P
- řešení: definovat programy tak, aby jejich důsledkem byly i negativní literály
zúplnění logického programu

Podstata zúplnění logického programu

- převod všech **if** příkazů v logickém programu na **iff**
 - $nad(X, Y) : \neg na(X, Y).$
 $nad(X, Y) : \neg na(X, Z), nad(Z, Y).$
 - lze psát jako: $nad(X, Y) : \neg (na(X, Y)) \vee (na(X, Z), nad(Z, Y)).$
 - zúplnění: $nad(X, Y) \leftrightarrow (na(X, Y)) \vee (na(X, Z), nad(Z, Y)).$

Podstata zúplnění logického programu

- převod všech **if** příkazů v logickém programu na **iff**
 - $nad(X, Y) : \neg na(X, Y)$.
 - $nad(X, Y) : \neg na(X, Z), nad(Z, Y)$.
 - lze psát jako: $nad(X, Y) : \neg (na(X, Y)) \vee (na(X, Z), nad(Z, Y))$.
 - zúplnění: $nad(X, Y) \leftrightarrow (na(X, Y)) \vee (na(X, Z), nad(Z, Y))$.
 - X je nad Y **právě tehdy, když alespoň jedna z podmínek platí**
 - tedy **pokud žádná z podmínek neplatí, X není nad Y**

Podstata zúplnění logického programu

- převod všech **if** příkazů v logickém programu na **iff**
 - $nad(X, Y) : \neg na(X, Y).$
 $nad(X, Y) : \neg na(X, Z), nad(Z, Y).$
 - lze psát jako: $nad(X, Y) : \neg (na(X, Y)) \vee (na(X, Z), nad(Z, Y)).$
 - zúplnění: $nad(X, Y) \leftrightarrow (na(X, Y)) \vee (na(X, Z), nad(Z, Y)).$
 - X je nad Y **právě tehdy, když alespoň jedna z podmínek platí**
 - tedy **pokud žádná z podmínek neplatí, X není nad Y**
- kombinace klauzulí je možná pouze pokud mají identické hlavy
 - $na(c, b).$
 $na(b, a).$
 - lze psát jako: $na(X_1, X_2) : \neg X_1 = c, X_2 = b.$
 $na(X_1, X_2) : \neg X_1 = b, X_2 = a.$
 - zúplnění: $na(X_1, X_2) \leftrightarrow (X_1 = c, X_2 = b) \vee (X_1 = b, X_2 = a).$

Zúplnění programu

- **Zúplnění programu** P je: $\text{comp}(P) := \text{IFF}(P) \cup \text{CET}$
- Základní vlastnosti:
 - $\text{comp}(P) \models P$
 - do programu je přidána pouze negativní informace

Zúplnění programu

- **Zúplnění programu** P je: $\text{comp}(P) := \text{IFF}(P) \cup \text{CET}$
- Základní vlastnosti:
 - $\text{comp}(P) \models P$
 - do programu je přidána pouze negativní informace
- **IFF(P)**: spojka $:-$ v $\text{IF}(P)$ je nahrazena spojkou \leftrightarrow
- **IF(P)**: množina všech formulí $\text{IF}(q, P)$
pro všechny predikátové symboly q v programu P
- Cíl: definovat $\text{IF}(q, P)$
- $\text{def}(p/n)$ predikátu p/n je množina všech klauzulí predikátu p/n

IF(q, P)

$$na(X_1, X_2) : \underbrace{-\exists Y(X_1 = c, X_2 = b, f(Y))}_{na(c, b) : -f(Y)} \vee (X_1 = b, X_2 = a, g).$$

● q/n predikátový symbol programu P $na(b, a) : -g.$

● X_1, \dots, X_n jsou „nové“ proměnné, které se nevyskytují nikde v P

● Necht' C je klauzule ve tvaru

$$q(t_1, \dots, t_n) : -L_1, \dots, L_m$$

kde $m \geq 0$, t_1, \dots, t_n jsou termy a L_1, \dots, L_m jsou literály.

Pak označme $E(C)$ výraz $\exists Y_1, \dots, Y_k(X_1 = t_1, \dots, X_n = t_n, L_1, \dots, L_m)$

kde Y_1, \dots, Y_k jsou všechny proměnné v C .

IF(q, P)

$$na(X_1, X_2) : \underline{-\exists Y(X_1 = c, X_2 = b, f(Y))} \vee (X_1 = b, X_2 = a, g).$$

● q/n predikátový symbol programu P $\underline{na(c, b) : -f(Y)}$. $na(b, a) : -g$.

● X_1, \dots, X_n jsou „nové“ proměnné, které se nevyskytují nikde v P

● Necht' C je klauzule ve tvaru

$$q(t_1, \dots, t_n) : -L_1, \dots, L_m$$

kde $m \geq 0$, t_1, \dots, t_n jsou termy a L_1, \dots, L_m jsou literály.

Pak označme $\mathbf{E}(C)$ výraz $\exists Y_1, \dots, Y_k(X_1 = t_1, \dots, X_n = t_n, L_1, \dots, L_m)$

kde Y_1, \dots, Y_k jsou všechny proměnné v C .

● Necht' $def(q/n) = \{C_1, \dots, C_j\}$.

Pak formuli $\mathbf{IF}(q, P)$ získáme následujícím postupem:

$$q(X_1, \dots, X_n) : -\mathbf{E}(C_1) \vee \mathbf{E}(C_2) \vee \dots \vee \mathbf{E}(C_j) \quad \text{pro } j > 0 \text{ a}$$

$$q(X_1, \dots, X_n) : -\square \quad \text{pro } j = 0 \text{ [} q/n \text{ není v programu } P \text{]}$$

Clarkova Teorie Rovnosti (CET)

všechny formule jsou univerzálně kvantifikovány:

1. $X = X$

2. $X = Y \rightarrow Y = X$

3. $X = Y \wedge Y = Z \rightarrow X = Z$

4. pro každý f/m : $X_1 = Y_1 \wedge \dots \wedge X_m = Y_m \rightarrow f(X_1, \dots, X_m) = f(Y_1, \dots, Y_m)$

5. pro každý p/m : $X_1 = Y_1 \wedge \dots \wedge X_m = Y_m \rightarrow (p(X_1, \dots, X_m) \rightarrow p(Y_1, \dots, Y_m))$

6. pro všechny různé f/m a g/n , ($m, n \geq 0$): $f(X_1, \dots, X_m) \neq g(Y_1, \dots, Y_n)$

7. pro každý f/m : $f(X_1, \dots, X_m) = f(Y_1, \dots, Y_m) \rightarrow X_1 = Y_1 \wedge \dots \wedge X_m = Y_m$

8. pro každý term t obsahující X jako vlastní podterm: $t \neq X$

$X \neq Y$ je zkrácený zápis $\neg(X = Y)$

Korektnost a úplnost NF pravidla

- **Korektnost NF pravidla:** Necht' P logický program a $: \neg A$ cíl.
Jestliže $: \neg A$ má definitivně neúspěšný SLD-strom,
pak $\forall (\neg A)$ je logickým důsledkem $\text{comp}(P)$ (nebo-li $\text{comp}(P) \models \forall (\neg A)$)

Korektnost a úplnost NF pravidla

- **Korektnost NF pravidla:** Necht' P logický program a $: \neg A$ cíl.
Jestliže $: \neg A$ má definitivně neúspěšný SLD-strom,
pak $\forall (\neg A)$ je logickým důsledkem $\text{comp}(P)$ (nebo-li $\text{comp}(P) \models \forall (\neg A)$)
- **Úplnost NF pravidla:** Necht' P je logický program. Jestliže $\text{comp}(P) \models \forall (\neg A)$,
pak existuje definitivně neúspěšný SLD-strom $: \neg A$.
 - zůstává problém: není rozhodnutelné, zda daná atomická formule je logickým důsledkem daného logického programu.
 - teorém mluví pouze o **existenci** definitivně neúspěšného SLD-stromu
 - definitivně (konečně) neúspěšný SLD-strom sice existuje, ale nemusíme ho nalézt
 - např. v Prologu: může existovat konečné odvození, ale program přesto cyklí (Prolog nenajde definitivně neúspěšný strom)

Korektnost a úplnost NF pravidla

- **Korektnost NF pravidla:** Necht' P logický program a $: \neg A$ cíl.
Jestliže $: \neg A$ má definitivně neúspěšný SLD-strom,
pak $\forall (\neg A)$ je logickým důsledkem $\text{comp}(P)$ (nebo-li $\text{comp}(P) \models \forall (\neg A)$)
- **Úplnost NF pravidla:** Necht' P je logický program. Jestliže $\text{comp}(P) \models \forall (\neg A)$,
pak existuje definitivně neúspěšný SLD-strom $: \neg A$.
 - zůstává problém: není rozhodnutelné, zda daná atomická formule je logickým důsledkem daného logického programu.
 - teorém mluví pouze o **existenci** definitivně neúspěšného SLD-stromu
 - definitivně (konečně) neúspěšný SLD-strom sice existuje, ale nemusíme ho nalézt
 - např. v Prologu: může existovat konečné odvození, ale program přesto cyklí (Prolog nenajde definitivně neúspěšný strom)
- Odvození pomocí NF pouze **test**, nelze **konstruovat** výslednou substituci
 - v $(\text{comp}(P) \models \forall (\neg A))$ je A všeob. kvantifikováno, v $\forall (\neg A)$ nejsou volné proměnné

Normální a stratifikované programy

- **normální program:** obsahuje negativní literály v pravidlech
- problém: existence zúplnění, která nemají žádný model
 - $p : -\neg p.$ zúplnění: $p \leftrightarrow \neg p$
- rozdělení programu na vrstvy
 - vynucují použití negace relace pouze tehdy pokud je relace úplně definovaná

Normální a stratifikované programy

• **normální program:** obsahuje negativní literály v pravidlech

• problém: existence zúplnění, která nemají žádný model

• $p : -\neg p.$ zúplnění: $p \leftrightarrow \neg p$

• rozdělení programu na vrstvy

• vynucují použití negace relace pouze tehdy pokud je relace úplně definovaná

• $a.$

$a : -\neg b, a.$

$b.$

$a.$

$a : -\neg b, a.$

$b : -\neg a.$

Normální a stratifikované programy

- **normální program:** obsahuje negativní literály v pravidlech
- problém: existence zúplnění, která nemají žádný model
 - $p : -\neg p.$ zúplnění: $p \leftrightarrow \neg p$
- rozdělení programu na vrstvy
 - vynucují použití negace relace pouze tehdy pokud je relace úplně definovaná
 - | | |
|-------------------|---------------------|
| $a.$ | $a.$ |
| $a : -\neg b, a.$ | $a : -\neg b, a.$ |
| $b.$ | $b : -\neg a.$ |
| stratifikovaný | není stratifikovaný |

Normální a stratifikované programy

- **normální program:** obsahuje negativní literály v pravidlech
- problém: existence zúplnění, která nemají žádný model
 - $p : -\neg p.$ zúplnění: $p \leftrightarrow \neg p$
- rozdělení programu na vrstvy
 - vynucují použití negace relace pouze tehdy pokud je relace úplně definovaná
 - | | |
|-------------------|---------------------|
| $a.$ | $a.$ |
| $a : -\neg b, a.$ | $a : -\neg b, a.$ |
| $b.$ | $b : -\neg a.$ |
| stratifikovaný | není stratifikovaný |
- normální program P je **stratifikovaný**: množina predikátových symbolů programu lze rozdělit do disjunktních množin S_0, \dots, S_m ($S_i \equiv$ **stratum**)
 - $p(\dots) : - \dots, q(\dots), \dots \in P, p \in S_k \implies q \in S_0 \cup \dots \cup S_k$
 - $p(\dots) : - \dots, \neg q(\dots), \dots \in P, p \in S_k \implies q \in S_0 \cup \dots \cup S_{k-1}$

Stratifikované programy II

- program je **m -stratifikovaný** $\iff m$ je nejmenší index takový, že $S_0 \cup \dots \cup S_m$ je množina všech predikátových symbolů z P
- **Věta:** Zúplnění každého stratifikovaného programu má Herbrandův model.
 - $p : -\neg p.$ nemá Herbrandův model
 - $p : -\neg p.$ ale není stratifikovaný

Stratifikované programy II

- program je **m -stratifikovaný** $\iff m$ je nejmenší index takový, že $S_0 \cup \dots \cup S_m$ je množina všech predikátových symbolů z P
- **Věta:** Zúplnění každého stratifikovaného programu má Herbrandův model.
 - $p : -\neg p.$ nemá Herbrandův model
 - $p : -\neg p.$ ale není stratifikovaný
- stratifikované programy nemusí mít **jedinečný** minimální Herbrandův model
 - $cykli : -\neg zastavi.$
 - dva minimální Herbrandovy modely: $\{cykli\}, \{zastavi\}$
 - důsledek toho, že $cykli : -\neg zastavi.$ je ekvivalentní $cykli \vee zastavi$

SLDNF rezoluce: úspěšné odvození

- NF pravidlo:
$$\frac{\vdash \neg C. \text{ má konečně neúspěšný SLD-strom}}{\vdash C}$$
- Pokud máme negativní podcíl $\neg C$ v dotazu G , pak hledáme důkaz pro C
- **Pokud odvození C selže (strom pro C je konečně neúspěšný), pak je odvození G (i $\neg C$) celkově úspěšné**

nahore(X) : $\neg \neg \text{blokovany}(X)$.

blokovany(X) : $\neg \text{na}(Y, X)$.

na(a, b).

SLDNF rezoluce: úspěšné odvození

- NF pravidlo:
$$\frac{:- C. \text{ má konečně neúspěšný SLD-strom}}{\neg C}$$
- Pokud máme negativní podcíl $\neg C$ v dotazu G , pak hledáme důkaz pro C
- Pokud odvození C selže (strom pro C je konečně neúspěšný), pak je odvození G (i $\neg C$) celkově úspěšné

$nahore(X) : \neg \neg blokovany(X).$

$blokovany(X) : \neg na(Y, X).$

$na(a, b).$

$:- nahore(c).$

yes

$$\begin{array}{c} :- nahore(c). \\ | \\ :- \neg blokovany(c). \end{array}$$

$:- blokovany(c).$

$$\begin{array}{c} | \\ :- na(Y, c). \end{array}$$

$$\begin{array}{c} | \\ FAIL \end{array}$$

\Rightarrow úspěšné odvození

SLDNF rezoluce: neúspěšné odvození

- NF pravidlo:
$$\frac{\neg C. \text{ má konečně neúspěšný SLD-strom}}{\neg C}$$
- Pokud máme negativní podcíl $\neg C$ v dotazu G , pak hledáme důkaz pro C
- Pokud existuje vyvrácení C s prázdnou substitucí (strom pro C je konečně úspěšný), pak je odvození G (i $\neg C$) celkově neúspěšné

nahore(X) : $\neg \neg \text{blokovany}(X)$.

blokovany(X) : $\neg \text{na}(Y, X)$.

na($_$, $_$).

SLDNF rezoluce: neúspěšné odvození

- NF pravidlo:
$$\frac{:- C. \text{ má konečně neúspěšný SLD-strom}}{\neg C}$$
- Pokud máme negativní podcíl $\neg C$ v dotazu G , pak hledáme důkaz pro C
- Pokud existuje vyvrácení C s prázdnou substitucí (strom pro C je konečně úspěšný), pak je odvození G (i $\neg C$) celkově neúspěšné

$nahore(X) : \neg \neg blokovany(X).$

$blokovany(X) : \neg na(Y, X).$

$na(., .).$

$:- nahore(X).$

no

$$\begin{array}{c} :- nahore(X). \\ | \\ :- \neg blokovany(X). \end{array}$$

$$\begin{array}{c} :- blokovany(X). \\ | \\ :- na(Y, X). \\ | \\ \square \end{array}$$

\Rightarrow **neúspěšné odvození**

SLDNF rezoluce: uvázlé odvození

- NF pravidlo:
$$\frac{\neg C \text{ má konečně neúspěšný SLD-strom}}{\neg C}$$
- Pokud máme negativní podcíl $\neg C$ v dotazu G , pak hledáme důkaz pro C
- Pokud existuje vyvrácení C s neprázdnou substitucí (strom pro C je konečně úspěšný), pak je odvození G (i $\neg C$) uvázlé

$nahore(X) : \neg \neg blokovany(X)$.

$blokovany(X) : \neg na(Y, X)$.

$na(a, b)$.

SLDNF rezoluce: uvázlé odvození

- NF pravidlo:
$$\frac{\vdash - C. \text{ má konečně neúspěšný SLD-strom}}{\vdash - C}$$
- Pokud máme negativní podcíl $\neg C$ v dotazu G , pak hledáme důkaz pro C
- Pokud existuje vyvrácení C s neprázdnou substitucí (strom pro C je konečně úspěšný), pak je odvození G (i $\neg C$) uvázlé

$nahore(X) : -\neg blokovany(X).$

$blokovany(X) : -na(Y, X).$

$na(a, b).$

$:- nahore(X).$

$$\begin{array}{c} :- nahore(X). \\ | \\ :- \neg blokovany(X). \end{array}$$

$:- blokovany(X).$

$$\begin{array}{c} | \\ :- na(Y, X). \\ | \quad [Y/a, X/b] \\ \square \\ [X/b] \end{array}$$

\Rightarrow uvázlé odvození

Cvičení: SLDNF odvození

Napište množinu SLDNF odvození pro uvedený dotaz.

$\text{:- } a(B).$

$a(X) \text{ :- } b(X), \text{ \+ } c(X).$

$a(X) \text{ :- } d(X), Y \text{ is } X+1, \text{ \+ } c(Y), b(X).$

$b(1).$

$c(A) \text{ :- } d(A).$

$d(1).$

SLD⁺ odvození

● P je normální program, G_0 normální cíl, R selekční pravidlo:

SLD⁺-odvození G_0 je buď konečná posloupnost

$$\langle G_0; C_0 \rangle, \dots, \langle G_{i-1}; C_{i-1} \rangle, G_i$$

nebo nekonečná posloupnost

$$\langle G_0; C_0 \rangle, \langle G_1; C_1 \rangle, \langle G_2; C_2 \rangle, \dots$$

kde v každém kroku $m + 1$ ($m \geq 0$), R vybírá **pozitivní literál** v G_m a dospívá k G_{m+1} obvyklým způsobem.

SLD⁺ odvození

- P je normální program, G_0 normální cíl, R selekční pravidlo:

SLD⁺-odvození G_0 je buď konečná posloupnost

$$\langle G_0; C_0 \rangle, \dots, \langle G_{i-1}; C_{i-1} \rangle, G_i$$

nebo nekonečná posloupnost

$$\langle G_0; C_0 \rangle, \langle G_1; C_1 \rangle, \langle G_2; C_2 \rangle, \dots$$

kde v každém kroku $m + 1$ ($m \geq 0$), R vybírá **pozitivní literál** v G_m a dospívá k G_{m+1} obvyklým způsobem.

- konečné SLD⁺-odvození může být:

1. **úspěšné:** $G_i = \square$

2. **neúspěšné**

3. **blokové:** G_i je negativní (např. $\neg A$)

SLDNF rezoluce: pojmy

● Úroveň cíle

- P normální program, G_0 normální cíl, R selekční pravidlo:

úroveň cíle G_0 se rovná

● $0 \iff$ žádné SLD^+ -odvození s pravidlem R není blokováno

● $k + 1 \iff$ maximální úroveň cílů : $\neg A$,

které ve tvaru $\neg A$ blokují SLD^+ -odvození G_0 , je k

- nekonečná úroveň cíle: **blokováné SLDNF odvození**

SLDNF rezoluce: pojmy

● Úroveň cíle

- P normální program, G_0 normální cíl, R selekční pravidlo:

úroveň cíle G_0 se rovná

● $0 \iff$ žádné SLD⁺-odvození s pravidlem R není blokováno

● $k + 1 \iff$ maximální úroveň cílů : $\neg A$,

které ve tvaru $\neg A$ blokují SLD⁺-odvození G_0 , je k

- nekonečná úroveň cíle: **blokováné SLDNF odvození**

● **Množina SLDNF odvození** = $\{(\text{SLDNF odvození } G_0) \cup (\text{SLDNF odvození : } \neg A)\}$

- při odvozování G_0 jsme se dostali k cíli $\neg A$

● SLDNF odvození cíle G ?

SLDNF rezoluce

P normální program, G_0 normální cíl, R selekční pravidlo:

množina SLDNF odvození a podmnožina neúspěšných SLDNF odvození cíle G_0 jsou takové nejmenší množiny, že:

- každé **SLD⁺-odvození** G_0 je SLDNF odvození G_0
- je-li SLD⁺-odvození $\langle G_0; C_0 \rangle, \dots, G_i$ **blokováno na $\neg A$**
 - tj. G_i je tvaru : $- L_1, \dots, L_{m-1}, \neg A, L_{m+1}, \dots, L_n$

pak

SLDNF rezoluce

P normální program, G_0 normální cíl, R selekční pravidlo:

množina SLDNF odvození a podmnožina neúspěšných SLDNF odvození cíle G_0 jsou takové nejmenší množiny, že:

- každé **SLD⁺-odvození** G_0 je SLDNF odvození G_0

- je-li SLD⁺-odvození $\langle G_0; C_0 \rangle, \dots, G_i$ **blokováno na $\neg A$**

 - tj. G_i je tvaru : $- L_1, \dots, L_{m-1}, \neg A, L_{m+1}, \dots, L_n$

pak

 - **existuje-li SLDNF odvození : $\neg A$** (pod R) s prázdnou cílovou substitucí, pak $\langle G_0; C_0 \rangle, \dots, G_i$ je **neúspěšné SLDNF odvození**

SLDNF rezoluce

P normální program, G_0 normální cíl, R selekční pravidlo:

množina SLDNF odvození a podmnožina neúspěšných SLDNF odvození cíle G_0 jsou takové nejmenší množiny, že:

- každé **SLD⁺-odvození** G_0 je SLDNF odvození G_0

- je-li SLD⁺-odvození $\langle G_0; C_0 \rangle, \dots, G_i$ **blokováno na $\neg A$**

- tj. G_i je tvaru : $- L_1, \dots, L_{m-1}, \neg A, L_{m+1}, \dots, L_n$

pak

- **existuje-li SLDNF odvození : $\neg A$** (pod R) s prázdnou cílovou substitucí, pak $\langle G_0; C_0 \rangle, \dots, G_i$ je **neúspěšné SLDNF odvození**

- je-li **každé úplné SLDNF odvození : $\neg A$ (pod R) neúspěšné** pak

- $\langle G_0; C_0 \rangle, \dots, \langle G_i, \epsilon \rangle, (: - L_1, \dots, L_{m-1}, L_{m+1}, \dots, L_n)$ je **(úspěšné) SLDNF odvození cíle G_0**

- ϵ označuje prázdnou cílovou substituci

Typy SLDNF odvození

Konečné SLDNF-odvození může být:

1. **úspěšné:** $G_i = \square$

2. **neúspěšné**

3. **uvázlé (flounder):**

G_i je negativní ($\neg A$) a $\vdash \neg A$ je úspěšné **s neprázdnou cílovou substitucí**

4. **blokové:** G_i je negativní ($\neg A$) a $\vdash \neg A$ nemá konečnou úroveň.

Korektnost a úplnost SLDNF odvození

● korektnost SLDNF-odvození:

P normální program, $: -G$ normální cíl a R je selekční pravidlo:

je-li θ cílová substituce SLDNF-odvození cíle $: -G$, pak

$G\theta$ je logickým důsledkem $\text{comp}(P)$

Korektnost a úplnost SLDNF odvození

● korektnost SLDNF-odvození:

P normální program, $: -G$ normální cíl a R je selekční pravidlo:

je-li θ cílová substituce SLDNF-odvození cíle $: -G$, pak

$G\theta$ je logickým důsledkem $\text{comp}(P)$

- implementace SLDNF v Prologu není korektní
- Prolog neřeší uvázané SLDNF-odvození (neprázdná substituce)
- použití bezpečných cílů (negace neobsahuje proměnné)

Korektnost a úplnost SLDNF odvození

● korektnost SLDNF-odvození:

P normální program, $: -G$ normální cíl a R je selekční pravidlo:

je-li θ cílová substituce SLDNF-odvození cíle $: -G$, pak

$G\theta$ je logickým důsledkem $\text{comp}(P)$

- implementace SLDNF v Prologu není korektní
- Prolog neřeší uvázané SLDNF-odvození (neprázdná substituce)
- použití bezpečných cílů (negace neobsahuje proměnné)

● úplnost SLDNF-odvození: SLDNF-odvození **není** úplné

● pokud existuje konečný neúspěšný strom $: -A$, pak $\neg A$ platí

ale místo toho se odvozování $: -A$ může zacyklit, tj. SLDNF rezoluce $\neg A$ neodvodí

$\Rightarrow \neg A$ tedy sice platí, ale SLDNF rezoluce ho nedokáže odvodit