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).
```

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

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

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

Cvičení: Napište minimální Herbrandův model pro následující logický program.

```
muz(petr). muz(pavel). zena(olga). 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 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 $P \cup \{G\}$.
  [1]tímto 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)

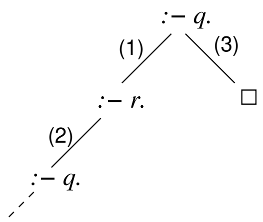
Výpočetní strategie

- **Korektní výpočetní strategie** prohledávání stromu výpočtu musí zaručit, že se každý (konečný) výsledek nalézt v konečném čase
- 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: $a(X,f(X)).$
  - by vedl k: $[B/X]$, $[X/f(X)]$
- Unifikátor pro $g(X_1,\ldots,X_n)$ a $g(f(X_0,X_0),f(X_1,X_1),\ldots,f(X_{n-1},X_{n-1}))$

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

  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 $X = f(f(f(f(f(f(f(f(f(f(\ldots))))))))))$

SLD-rezoluce v Prologu: korektnost

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

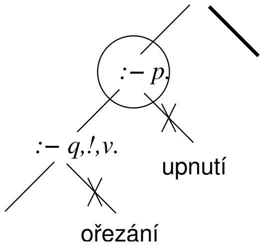
  $\Longrightarrow$ **není korektní**

(1) $t(X):-p(X,X).$ $:-t(X).$
$p(X,f(X)).$ $X = f(f(f(f(\ldots)))))))))$ 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"

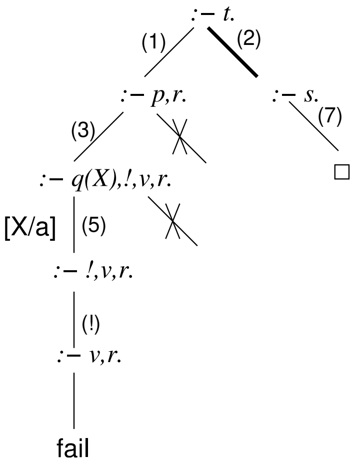
Řízení implementace: řez



- 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

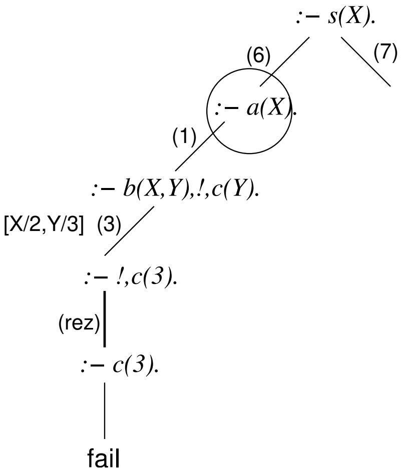
Příklad: řez

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



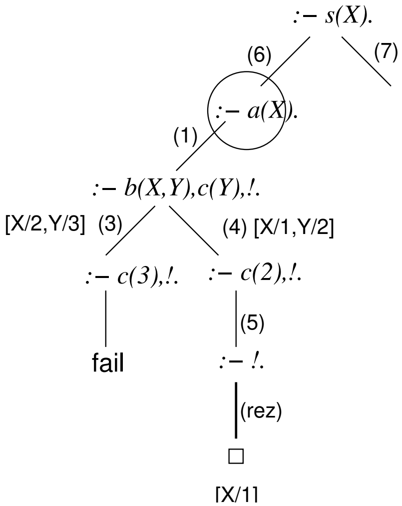
Příklad: řez II

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



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
- relace vyjádřeny explicitně v nejmenším Herbrandově modelu
  - $nad(X,Y) :- na(X,Y).$ $na(c,b).$
    $nad(X,Y) :- na(X,Z), nad(Z,Y).$ $na(b,a).$
  - 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 vyjadřena 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): $\dfrac{P \not\models A}{\neg A}$ (CWA)
- pro SLD-rezoluci: $P \not\models nad(a,c)$, 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 $:- A$
  $\dfrac{:- A \text{ má definitivně (konečně) neúspěšný SLD-strom}}{\neg A}$ **(*negation as failure*, NF)**
- **normální cíl**: cíl obsahující i negativní literály
  - $:- nad(c,a), \neg nad(b,c).$
- rozdíl mezi CWA a NF
  - program $nad(X,Y) :- nad(X,Y)$, cíl $:- \neg nad(b,c)$
  - neexistuje odvození cíle podle NF, protože SLD-strom $:- nad(b,c)$ je nekonečný
  - existuje odvození cíle podle CWA, protože neexistuje vyvrácení $:- nad(b,c)$
- 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**
  - $nad(X,Y) :- na(X,Y).$
    $nad(X,Y) :- na(X,Z), nad(Z,Y).$
  - lze psát jako: $nad(X,Y) :- (na(X,Y)) \vee (na(X,Z), nad(Z,Y)).$
  - 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: $na(X_1,X_2) :- X_1 = c, X_2 = b.$
    $na(X_1,X_2) :- X_1 = b, X_2 = a.$
  - 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: $\mathrm{comp}(P) := \mathrm{IFF}(P) \cup \mathrm{CET}$
- Základní vlastnosti:
  - $\mathrm{comp(P)} \models \mathrm{P}$
  - do programu je přidána pouze negativní informace
- $\mathbf{IFF(P)}$: spojka $:-$ v $\mathrm{IF(P)}$ je nahrazena spojkou $\leftrightarrow$
- $\mathbf{IF}(P)$: množina všech formulí $\mathbf{IF}(q,P)$
  pro všechny predikátové symboly q v programu P
- Cíl: definovat $\mathbf{IF}(q,P)$
- $def(p/n)$ predikátu p/n je množina všech klauzulí predikátu p/n

$\mathbf{IF}(q,P)$

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

- q/n predikátový symbol programu P
- $X_1, \ldots, X_n$ jsou „nové" proměnné, které se nevyskytují nikde v P
- Nechť C je klauzule ve tvaru
  $q(t_1, \ldots, t_n) :- L_1, \ldots, L_m$
  kde $m \geq 0$, $t_1, \ldots, t_n$ jsou termy a $L_1, \ldots, L_m$ jsou literály.

  Pak označme $\mathbf{E}(C)$ výraz $\exists Y_1, \ldots, Y_k(X_1 = t_1, \ldots, X_n = t_n, L_1, \ldots, L_m)$
  kde $Y_1, \ldots, Y_k$ jsou všechny proměnné v C.
- Nechť $def(q/n) = \{C_1, \ldots, C_j\}$.

  Pak formuli $\mathbf{IF}(q,P)$ získáme následujícím postupem:
  $q(X_1, \ldots, X_n) :- \mathbf{E}(C_1) \vee \mathbf{E}(C_2) \vee \cdots \vee \mathbf{E}(C_j)$ pro $j > 0$ a
  $q(X_1, \ldots, X_n) :- \Box$ pro $j = 0$ [q/n není v programu P]

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: $X_1 = Y_1 \wedge \cdots \wedge X_m = Y_m \rightarrow f(X_1, \ldots, X_m) = f(Y_1, \ldots, Y_m)$
5. pro každý p/m: $X_1 = Y_1 \wedge \cdots \wedge X_m = Y_m \rightarrow (p(X_1, \ldots, X_m) \rightarrow p(Y_1, \ldots, Y_m))$
6. pro všechny různé f/m a g/n, $(m, n \geq 0)$: $f(X_1, \ldots, X_m) \neq g(Y_1, \ldots, Y_n)$
7. pro každý f/m: $f(X_1, \ldots, X_m) = f(Y_1, \ldots, Y_m) \rightarrow X_1 = Y_1 \wedge \cdots \wedge X_m = Y_m$
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**: Nechť P logický program a $: -A$ cíl.
  Jestliže $: -A$ má definitivně neúspěšný SLD-strom,
  pak $\forall(\neg A)$ je logickým důsledkem $\mathrm{comp}(P)$ (nebo-li $\mathrm{comp}(P) \models \forall(\neg A)$)
- **Úplnost NF pravidla**: Nechť P je logický program. Jestliže $\mathrm{comp}(P) \models \forall(\neg A)$, pak existuje definitivně neúspěšný SLD-strom $: -A$.
  - 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 $(\mathrm{comp}(P) \models \forall(\neg A))$ 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
$a.$	$a.$
$a : -\neg b, a.$	$a : -\neg b, a.$
$b.$	$b : -\neg a.$
stratifikovaný	není stratifikovaný
- normální program P je **stratifikovaný**: množina predikátových symbolů programu lze rozdělit do disjunktních množin $S_0, \ldots, S_m$ ($S_i \equiv$ **stratum**)
  - $p(\ldots) : -\ldots, q(\ldots), \ldots \in P,\ p \in S_k \Longrightarrow q \in S_0 \cup \ldots \cup S_k$
  - $p(\ldots) : -\ldots, \neg q(\ldots), \ldots \in P,\ p \in S_k \Longrightarrow q \in S_0 \cup \ldots \cup S_{k-1}$

Stratifikované programy II

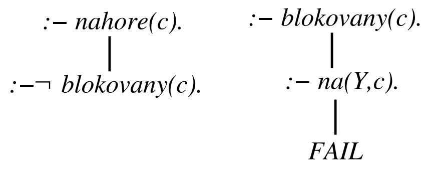
- program je m-**stratifikovaný** $\Longleftrightarrow$ m je nejmenší index takový, že $S_0 \cup \ldots \cup S_m$ je množina všech predikátových symbolů z P
- **Věta:** Zúplnění každého stratifikovaného programu má Herbrandův model.
  - $p : -\neg p.$ nemá Herbrandův model
  - $p : -\neg p.$ ale není 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: $\dfrac{: -C.\ \text{má konečně neúspěšný SLD-strom}}{\neg C}$
- 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**

$nahore(X) : -\neg blokovany(X).$
$blokovany(X) : -na(Y, X).$
$na(a, b).$

$: -nahore(c).$
yes



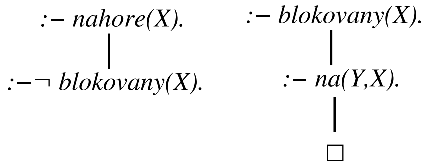
$\Rightarrow$ **uspěšné odvození**

SLDNF rezoluce: neúspěšné odvození

- NF pravidlo: $\dfrac{:-C.\ \text{má konečně neúspěšný SLD-strom}}{\neg C}$
- 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**

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

$:- nahore(X).$
no



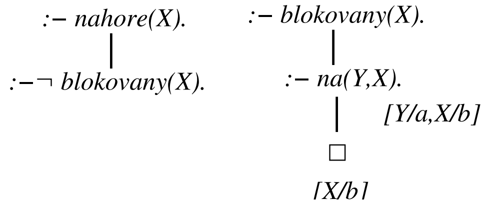
⇒ **neúspěšné odvození**

SLDNF rezoluce: uvázlé odvození

- NF pravidlo: $\dfrac{:-C.\ \text{má konečně neúspěšný SLD-strom}}{\neg C}$
- 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**

$nahore(X) :- \neg blokovany(X).$
$blokovany(X) :- na(Y,X).$
$na(a,b).$

$:- nahore(X).$



⇒ **uvázlé odvození**

Cvičení: SLDNF odvození

Napište množinu SLDNF odvození pro uvedený dotaz.

```
:- a(B).

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

b(1).

c(A) :- 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

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

  nebo nekonečná posloupnost

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

  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. **blokované:** 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
    - $\mathbf{0} \iff$ žádné SLD$^+$-odvození s pravidlem R není blokováno
    - $k+\mathbf{1} \iff$ maximální úroveň cílů $: -A$,
      které ve tvaru $\neg A$ blokují SLD$^+$-odvození G_0, je k
  - nekonečná úroveň cíle: **blokované SLDNF odvození**
- **Množina SLDNF odvození** = {(SLDNF odvození G_0) ∪ (SLDNF odvození $: -A$)}
  - 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 $\langle G_0; C_0 \rangle, \ldots, G_i$ **blokováno na** $\neg A$
  - tj. G_i je tvaru $: -\ L_1, \ldots, L_{m-1}, \neg A, L_{m+1}, \ldots, L_n$

  pak
  - **existuje-li SLDNF odvození** $: -A$ (pod R) s prázdnou cílovou substitucí, pak $\langle G_0; C_0 \rangle, \ldots, G_i$ je **neúspěšné SLDNF odvození**
  - je-li **každé úplné SLDNF odvození** $: -A$ **(pod** R**) neúspěšné** pak
    $\langle G_0; C_0 \rangle, \ldots, \langle G_i, \epsilon \rangle, (: -\ L_1, \ldots, L_{m-1}, L_{m+1}, \ldots, L_n)$ 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 $: -A$ je úspěšné **s neprázdnou cílovou substitucí**
4. **blokované:** G_i je negativní ($\neg A$) a $: -A$ 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 comp(P)
  - implementace SLDNF v Prologu není korektní
  - Prolog neřeší uvázlé 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

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