Paradigmata programování IV
Logické programování

Vilém VYCHODIL
KI PřF UP Olomouc

12. května 2005

Logické programování

One of the main ideas of **logic programming**, which is due to Kowalski, is that an algorithm consists of two disjoint components, the **logic** and the **control**. The logic is the statement **what** the problem is that has to be solved. The control is the statement of **how** it is to be solved. Generally speaking, a logic programming system should provide ways for the programmer to specify each of these components.

— John Wylie Lloyd: *Foundations of Logic Programming*

A **logic program** is a set of axioms, or rules, defining relationships between objects. A **computation** of a logic program is a deduction of consequences of the program. A program defines a set of consequences, which is its **meaning**. The **art of logic programming** is constructing concise and elegant programs that have the desired meaning.

— Leon Sterling and Ehud Shapiro: *The Art of Prolog*

Logické programování a jeho vývoj

Co je logické programování?

- paradigma programování
- založeno na matematické logice (automatické dokazování)
- **program = konečná množina axiomů**
- **výpočet = konstruktivní důkaz cíle (dotazu), který zadá uživatel**

Vývoj a použití

- princip **rezoluce**, představen ve článku:
  Robinson J. A.: A machine-oriented logic based on the resolution principle.
  Journal ACM **12**(1965), 23–41
- implementace: zač. 70. let 20. stol., Univerzita v Marseille: **PROLOG**
- programování pro **umělou inteligenci** (expertní systémy):
  dialekty LISPu (Common LISP, Scheme, . . .) v USA, PROLOG v Evropě
- rozšíření
  - fuzzy PROLOG (rozšíření o neurčitost – stupně pravdivosti)
  - DATALOG (dotazování v relačních databázích)
  - Prolog založený na lineární logice
    . . .

Logické programovací jazyky v praxi

Implementace

- překladače PROLOGu pracují **interpretačním způsobem**
- interakce s uživatelem:
  1. fáze: uživatel **načte program** v PROLOGu
  2. fáze: uživatel **zadává cíle** (dotazy) a PROLOG na ně odpovídá
- příklady: mnoho interpretů (SWI-Prolog, GNU Prolog, . . .)
- jednoduchý interpret PROLOGu lze snadno implementovat

Interprety používané na cvičení

- **SWI-Prolog**
  - autor Jan Wielemaker, `jan@swi.psy.uva.nl`, University of Amsterdam, `http://www.swi-prolog.org` (verze pro Unix, MS-Windows)
  - plně implementuje ISO PROLOG (standard) + další rozšíření (GUI, . . .)
- **scm-prolog**
  - autor Vilém Vychodil, `vilem.vychodil@upol.cz`, Palacký University `http://vychodil.inf.upol.cz` (spustitelné ve R^5RS Scheme)
  - pedagogická implementace PROLOGu ve Scheme (169 řádků)

Znalostní báze versus logický program

Znalostní báze (popis přirozeným jazykem)

„*Honzík, Jirka a Vilík jsou muži. Monika a Jana jsou ženy.*
Honzík je Jirkovo dítě. Vilík je Moničino dítě.
Synové jsou děti mužského pohlaví.
Děti jsou (naši) potomci, děti našich potomků jsou rovněž (naši) potomci.“

Přesnější reformulace znalostní báze (vhodná pro formalizaci)

„**Honzík**, **Jirka** *a* **Vilík jsou muži**. **Monika** *a* **Jana jsou ženy**.
Honzík je Jirkovo dítě. **Vilík je Moničino dítě**.
Člověk X je synem člověka Y *pokud* **je X dítětem Y** *a pokud* **je X muž**.
Člověk X je potomkem člověka Y *pokud* **je X dítětem Y** *nebo pokud* **je X dítětem některého potomka Y**.“

Formalizovaná znalostní báze (jednoduchý logický program)

```
male(john). male(george). male(bill).
female(monica). female(jane).
isChildOf(john,george). isChildOf(bill,monica).
isSonOf(X,Y) :- isChildOf(X,Y), male(X).
isOffspringOf(X,Y) :- isChildOf(X,Y).
isOffspringOf(X,Y) :- isChildOf(X,Z), isOffspringOf(Z,Y).
```

Motivační logický program

Ukázkový program (syntaxe odpovídá PROLOGu)

```
male(john). male(george). male(bill). male(harold).
female(monica). female(jane).

isChildOf(john,george).
isChildOf(george,bill).
isChildOf(bill,monica).
isChildOf(jane,george).

isSonOf(X,Y) :- isChildOf(X,Y), male(X).

isDaughterOf(X,Y) :- isChildOf(X,Y), female(X).

isOffspringOf(X,Y) :- isChildOf(X,Y).
isOffspringOf(X,Y) :- isChildOf(X,Z), isOffspringOf(Z,Y).
```

Motivační logický program

Uživatelské dotazy (prologovská interakce s uživatelem)

```
?- male(john). ENTER
Yes.

?- female(john). ENTER
No.

?- male(X). ENTER
X=john. ENTER
Yes.

?- isSonOf(john,george). ENTER
Yes.

?- female(X). ENTER
X=monica ;
X=jane ;
No.
```

Základní rysy logického programování

Program a výpočetní proces

- **logický program** je konečná množina formulí
  (tvrzení popisujících *modelovanou realitu*; formule mají *speciální tvar*)
- **výpočet** je zahájen zadáním formule–dotazu (tu zadává *uživatel*)
- **cílem výpočtu je najít důkaz** potvrzující, že dotaz **logicky vyplývá**
  (je dokazatelný) z logického programu (konstruktivnost)

Průběh a ukončení výpočetního procesu

- během výpočtu se (interpret) snaží ověřit **logické vyplývání**
- pokud je zjištěno, že dotaz z programu vyplývá, výpočet končí a uživateli je oznámeno „`Yes`“ s hodnotami případných proměnných, které se v dotazu vyskytují (hodnoty proměnných jsou *odpovědi*)
- dotaz může mít *víc jak jedno řešení*, většina interpretů je postupně nabízí (řešení může být někdy i *nekonečně mnoho*)
- pokud není zjištěno, že dotaz z programu vyplývá nebo pokud již žádné další řešení neexistuje, výpočet končí a uživateli je oznámeno „`No`“
- může se stát, že výpočet neskočí

Logické programování jako paradigma

Srovnání nejběžnějších paradigmat

- **procedurální:** výpočet = *sekvenční provádění procedur*
  významný rys: *přiřazovací příkaz*
  teoretický model: *RAM stroj (John von Neumann)*
- **funkcionální:** výpočet = *postupná aplikace funkcí*
  významný rys: *funkce vyšších řádů*
  teoretický model: *λ-kalkul (Alonzo Church)*
- **logické:** výpočet = *automatická dedukce*
  významný rys: *deklarativní charakter*
  teoretický model: *matematická logika, princip rezoluce (Robinson)*

Následující pojmy je třeba od sebe rozlišovat

- **čisté logické programování**
  (plně vysvětlitelné termíny matematické logiky)
- **logické programování**
  (čisté LP + prvky zvyšující komfort programátora)
- **(čistý) PROLOG**
  (konkrétní programovací jazyk založený na principech LP)
- **implementace PROLOGu**
  (překladač PROLOGu, z důvodu efektivity obvykle není „ideálně čistý“)

Odlišnost od ostatních paradigmat programování

Základní rysy logického programování

- programátor specifikuje, **co** *se má vypočítat*,
  nikoliv **jak** *se to má vypočítat a* **kam** *uložit mezivýsledky*
- **řízení výpočtu** programátor **neovlivňuje**: logické jazyky obvykle nemají příkazy pro řízení běhu výpočtu ani pro řízení toku dat, nemá příkazy cyklů, větvení, přiřazovací příkaz
- **proměnné**: neexistuje rozdělení proměnných na **vstupní** a **výstupní**, proměnná může být jednou použita jako vstupní, jindy jako výstupní; proměnná v PROLOGu označuje během výpočtu objekt (element, individuum), který vyhovuje jistým podmínkám
- nerozlišuje se mezi **daty a programem**.

Rysy překladačů logických jazyků

– ve většině (překladačů) logických jazyků jsou k dispozici řídící konstrukce
– je třeba mít na paměti, že **programátor není zbaven zodpovědnosti** za to, jak bude výpočet probíhat; výpočet je řízen primárně samotným překladačem a programátor musí znát pravidla, kterými se výpočet řídí, a v souladu s nimi programy vytvářet

Jazyk logického programu

různé logické programy = různé *probémové domény* = různé *elementy a vztahy*

```
male(john). male(george).
female(monica).
isChildOf(john,george).
isSonOf(X,Y) :- isChildOf(X,Y),
      male(X).
```

```
even(zero).
even(succ(succ(X))) :- even(X).
plus(X,zero,X).
plus(X,succ(Y),succ(Z)) :-
      plus(X,Y,Z).
```

Jazyk logického programu $\mathcal{L}$ je určen

(i) množinou F **funkčních symbolů** (**funktorů**, jména *funkčních závislostí*)
(ii) množinou R **relačních symbolů** (**predikátů**, jména *vlastností a vztahů*)

spolu s definicí **arity** (**počtu argumentů**) všech symbolů.

Značení symbolů a arity

- obecně povolujeme $F \cap R \neq \emptyset$ (srovnej s typem jazyka PL)
- aritu symbolů zapusujeme jako *číslo za symbolem oddělené* „/“ například $john/0$, $male/1$, $succ/1$, $is\_child\_of/2$, $plus/3$, . . .
- relační a funkční symboly vždy **začínají malým písmenem**
- nulární funktory (funktory tvaru $c/0$) nazýváme **konstanty** (**atomy**)
- pro každý logický program P uvažujeme **minimálni jazyk** $\mathcal{L}_P$ (nejmenší jazyk, ve kterém lze program P zapsat)

Minimální jazyk logického programu

Příklad 1.

```
male(john). male(george).
female(monica).
isChildOf(john,george).
isSonOf(X,Y) :- isChildOf(X,Y), male(X).
```

funkční symboly: $john/0$, $george/0$, $monica/0$
relační symboly: $male/1$, $female/1$, $isChildOf/2$, $isSonOf/2$

Příklad 2.

```
even(zero).
even(succ(succ(X))) :- even(X).
plus(X,zero,X).
plus(X,succ(Y),succ(Z)) :- plus(X,Y,Z).
mult(Whatever,zero,zero).
mult(X,succ(Y),Z) :- mult(X,Y,R), plus(X,R,Z).
```

funkční symboly: $zero/0$, $succ/1$
relační symboly: $even/1$, $plus/3$, $mult/3$

Termy a atomické formule

Proměnné – mohou na ně být **navázány hodnoty** (hodnoty = termy)

- značení: proměnné začínají **velkým písmenem** (!)
- příklad: X, Xyz, $Variable$ (v PROLOGu mohou začínat podtržítkem „_“)

Máme-li dán **jazyk $\mathcal{L}$**, zavádíme pojmy term a atomická formule jako v PL:

Termy jazyka $\mathcal{L}$:

(i) každá proměnná je term;
(ii) každá konstanta (atom) je term;
(iii) jsou-li $t_1, \ldots, t_n$ termy a f/n je funktor, pak $f(t_1, \ldots, t_n)$ je term

Bod (ii) je de facto nadbytečný, protože konstany jsou funktory $c/0$.

Atomická formule jazyka $\mathcal{L}$:

jsou-li $t_1, \ldots, t_n$ termy a r/n je relační symbol,
pak $r(t_1, \ldots, t_n)$ je atomická formule.

Uzavřené termy a formule

- term t je **uzavřený**, pokud neobsahuje žádnou proměnnou
- formule p je **uzavřená**, pokud neobsahuje žádnou proměnnou

Fakty a pravidla

Fakty a pravidla = základní stavební kameny programů v PROLOGu

Fakt = popis základního vztahu (mezi elementy)
Každá **atomická formule** je **fakt**.

Příklad:

$male(john)$	`male(john).`
$plus(Num, zero, Num)$	`plus(Num,zero,Num).`
$lessOrEqual(X, succ(X))$	`lessOrEqual(X,succ(X)).`

V PROLOGu se před „(“ *nesmí psát mezera*; každý fakt je ukončen *tečkou.*

Pravidlo = popis vztahů „jestliže . . . pak“ mezi fakty
Pravidla jsou výrazy tvaru $A_0 \leftarrow A_1, \ldots, A_n$ $(n \in \mathbb{N})$, kde
$A_0, \ldots, A_n$ jsou **atomické formule**.

- intuitivní význam $A_0 \leftarrow A_1, \ldots, A_n$: „pokud $A_1, \ldots, A_n$ pak A_0“
- A_0 se nazývá **hlava pravidla**
- $A_1, \ldots, A_n$ se nazývá **tělo pravidla**

Příklad:

$plus(X, succ(Y), succ(Z)) \leftarrow plus(X, Y, Z)$

```
plus(X,succ(Y),succ(Z)) :- plus(X,Y,Z).
```

$isSonOf(X, Y) \leftarrow isChildOf(X, Y), male(X)$

```
isSonOf(X,Y) :- isChildOf(X,Y), male(X).
```

Definitní program

Definitní klauzule = sjednocující pohled na fakty a pravidla

Každý výraz tvaru $A_0 \leftarrow A_1, \ldots, A_n$ $(n \in \mathbb{N}_0)$,
kde $A_0, \ldots, A_n$ jsou atomické formule, nazveme **definitní klauzule**.

Speciálně:

- pro $n \geq 1$ je tělo $A_0 \leftarrow A_1, \ldots, A_n$ neprázdné (**pravidlo**)
- pro $n = 0$ je $A_0 \leftarrow A_1, \ldots, A_n$ ve tvaru $A_0 \leftarrow$ (**fakt**)

Poznámky

- fakty (pravidla) jsou definitní klauzule s prázdným (neprázdným) tělem
- fakty značíme obvykle jen A_0 místo $A_0 \leftarrow$
- *hlava definitní klazule* je *vždy neprázdná*

Definitní program = konečná množina definitních klauzulí

- **definitní program** je speciální (jednoduchý) logický program
- na definitní programy se lze dívat jako na **formalizaci báze znalostí**
- definitními programy lze vyjádřit pouze „pozitivní znalost“
  (popisujeme platnost vztahů, nelze popsat neplatnost vztahů)

Proměnné v definitních klauzulích

Účel proměnných v klauzulích

- proměnné označují **nespecifikované elementy**
- proměnné mají **univerzální charakter**
- principiálně **jiný pohled** než u většiny **ostatních paradigmat**
  (proměnné neoznačují paměťová místa pro „ukládání hodnot“)

Sdílení proměnných

- proměnné se **sdílejí v rámci jedné klauzule**
- proměnné se **nesdílejí v rámci různých klauzulí**

Příklad: $isSonOf(X, Y) \leftarrow isChildOf(X, Y), male(X)$ ... X, Y sdílené v hlavě i těle

$$\left.\begin{array}{l} isOffspringOf(\boldsymbol{X}, Z) \leftarrow isChildOf(\boldsymbol{X}, Z) \\ isOffspringOf(\boldsymbol{X}, R) \leftarrow isChildOf(\boldsymbol{X}, Y), isOffspringOf(Y, R) \end{array}\right\} \text{zde } \boldsymbol{X} \neq \boldsymbol{X}$$

Logický pohled na klauzule a programy

- definitní klauzule jsou *uzavřené*, *univerzálně kvantifikované* formule
  $A_0 \leftarrow A_1, \ldots, A_n$ je formule tvaru $(\forall X_1) \cdots (\forall X_k)((A_1 \wedge \cdots \wedge A_n) \rightarrow A_0)$,
  kde $x_1, \ldots, x_k$ jsou všechny proměnné vyskytující se ve formulích $A_0, \ldots, A_n$
- logické programy jsou teorie obsahující speciální axiomy

Substituce a jejich aplikace

Substituce = **navázání hodnoty na proměnné**

Definice. Nechť $X_1, \ldots, X_n$ jsou proměnné a $t_1, \ldots, t_n$ jsou termy, kde

(i) $X_i \neq t_i$ $(i = 1, \ldots, n)$,
(ii) $X_i \neq X_j$ $(i \neq j)$,

pak množinu $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ nazveme **substituce**.

Příklad: $\theta_1 = \{X/john, Y/Z, Z/bill\}$, $\theta_2 = \{X/succ(succ(Y)), Z/zero\}$, $\theta_3 = \{\}$

Aplikace substituce = **souběžná náhrada proměnných termy**
Použití (**aplikace**) substituce θ na term (formuli/klauzuli) A je term (formule/ /klauzule) $A\theta$, který vznikne z A **souběžnou** (**paralelní**) **náhradou** všech výskytů promenných X_i v A příslušnými termy t_i.

Příklady: pro $\theta = \{X/john, Y/Z, Z/bill\}$ máme
1. $A = male(john)$; $A\theta = male(john)$
2. $A = male(Z)$; $A\theta = male(bill)$
3. $A = male(Y)$; $A\theta = male(Z)$ (sekvenční aplikací bychom dostali $male(bill)$)
4. $A = isOffspringOf(X, Y) \leftarrow isChildOf(X, Z), isOffspringOf(Z, Y)$;
   $A\theta = isOffspringOf(john, Z) \leftarrow isChildOf(john, bill), isOffspringOf(bill, Z)$
   $A\theta \neq isOffspringOf(john, bill) \leftarrow isChildOf(john, bill), isOffspringOf(bill, bill)$ (!)

Vazby, instance a uzavřené instance

Neformálně: **instance A** = „speciální případ A“ po provedení substituce

Definice (instance).
B je instancí A, pokud existuje substituce θ taková, že $B = A\theta$.

Příklad: $male(john)$ je instancí $male(X)$
$male(Y)$ je instancí $male(X)$
$isOffspringOf(john, george) \leftarrow isChildOf(john, X), isOffspringOf(X, george)$
je instancí $isOffspringOf(X, Y) \leftarrow isChildOf(X, Z), isOffspringOf(Z, Y)$

Definice (uzavřená instance).
Pokud je A instancí B a A je **uzavřená** (neobsahuje proměnné), pak se A nazývá **uzavřená instance B**. Je-li P definitní program, pak $\mathrm{ground}(P)$ označuje množinu všech uzavřených instancí všech definitních klauzulí z programu P.

Příklad: $male(john)$ je uzavřená instance $male(X)$
$male(Y)$ není uzavřená instance $male(X)$
$even(succ(succ(succ(zero)))) \leftarrow even(succ(zero))$
je uzavřená instance $even(succ(succ(X))) \leftarrow even(X)$

Poznámka. $(A_0 \leftarrow A_1, \ldots, A_n)\theta = A_0\theta \leftarrow A_1\theta, \ldots, A_n\theta$

Anonymní proměnné

Anonymní proměnné (singletony) – rozšíření poskytované PROLOGem

- speciální proměnná značená „_“ (podtržítko)
- anonymní proměnná se **nesdílí v rámci pravidel**
- na anonymní proměnnou se **nelze odkazovat**
- neformálně: anonymní proměnná je proměnná na jejíž „vazbě“ nezáleží

V čistém čistém logickém programování se anonymní proměnné **nezavádějí**, protože jsou plně **definovatelné**.

Definice. Proměnná X se nazývá **anonymní v** $A_0 \leftarrow A_1, \ldots, A_n$, pokud má X ve všech atomických formulích $\{A_0, \ldots, A_n\}$ dohromady právě jeden výskyt.

Příklad: $son(X) \leftarrow isChildOf(X, Y), male(X)$ . . . X není anonymní, Y je anonymní

Poznámka. Pokud překladač PROLOGu nemá anonymní proměnnou, pak bychom téhož efektu jako má „_“ dosáhli tím, že *každý výskyt „_“ v definitní klauzuli nahradíme novou proměnnou, která se dosud v klauzuli nevyskytuje.*

$son(X) \leftarrow isChildOf(X, \_), male(X)$ nahradíme $son(X) \leftarrow isChildOf(X, Y), male(X)$
$prop(X, \_) \leftarrow mult(\_, \_, X)$ nahradíme $prop(X, Y_1) \leftarrow mult(Y_2, Y_3, X)$

Pozor na **podstatný rozdíl**: $prop(X, X)$ versus $prop(\_, \_)$. (!)

Definitní cíl

Neformálně: **cíl/dotaz** = uživatelem zadaná otázka, spouští výpočet

Definitní cíl je výraz tvaru $\leftarrow A_1, \ldots, A_n$ $(n \in \mathbb{N}_0)$,
kde $A_1, \ldots, A_n$ jsou atomické formule

- proměnné se **sdílejí v celém cíli**
- cíle/dotazy mají **existenční charakter**
- **prázdný cíl** (pro $n = 0$) značíme $\square$ (zavádíme z technických důvodů)

Příklad:	$\leftarrow male(john)$	`?- male(john).`
	$\leftarrow male(X)$	`?- male(X).`
	$\leftarrow plus(succ(zero), Y, Z), even(Z)$	`?- plus(succ(zero),Y,Z), even(Z).`
	$\leftarrow isOffspringOf(john, X), male(X)$	`?- isOffspringOf(john,X),male(X).`

Slovně: „Platí, že $john$ je má vlastnost $male$?“
„Existují nějaké atomy s vlastností $male$?“, . . .

Shrnutí

- **definitní program** = logický program (množina definitních klauzulí)
- **definitní cíl** = uživatelský dotaz (definitní cíl **není** definitní klauzule)

Definitní programy a cíle jako klauzule

- **klauzule** $B_1, \ldots, B_m \leftarrow A_1, \ldots, A_n$ $(m, n \in \mathbb{N}_0)$
  - **hornovská klauzule** $B_1 \leftarrow A_1, \ldots, A_n$ $(m \in \{0, 1\},\ n \in \mathbb{N}_0)$
    - **definitní klauzule** $B_1 \leftarrow A_1, \ldots, A_n$ $(m = 1,\ n \in \mathbb{N}_0)$
      - **pravidlo** $B_1 \leftarrow A_1, \ldots, A_n$ $(m = 1,\ n \in \mathbb{N})$
      - **fakt** $B_1 \leftarrow$ $(m = 1,\ n = 0)$
    - **definitní cíl** $\leftarrow A_1, \ldots, A_n$ $(m = 0,\ n \in \mathbb{N}_0)$
      - **prázdný cíl** $\square$ $(m = 0,\ n = 0)$

Sémantika logického programu

Logický program nemá sám o sobě žádný význam (srovnej: pojem teorie v PL).

Deklarativní sémantika $A_0 \leftarrow A_1, \ldots, A_n$:

„Pokud platí A_1 a $\cdots$ a A_n, pak platí A_0.“

- význam programu a vyplývání z programu je založeno na **pojmu model**
- sémantika nezávislá na:
  - pořadí klauzulí v programu
  - pořadí atomických formulí v klauzulích

Operační (procedurální) sémantika $A_0 \leftarrow A_1, \ldots, A_n$:

„K tomu abychom vypočetli A_0 stačí postupně vypočíst $A_1, \ldots, A_n$.“

- význam programu a vyplývání je založeno na **postupné inferenci**
- ve své *čisté podobě* (čisté logické programování) je rovněž nezávislá na pořadí klauzulí v programu / atomických formulí v klauzulích
- **operační sémantika implementací PROLOGu** však již **závisí na pořadí**

Ideální stav – obě sémantiky splývají

– teoreticky lze / implementačně nedosažitelné

Herbrandovo universum, Herbrandova báze

Vstupní informace:

P . . . definitní program

$\mathcal{L}_P$ . . . jazyk programu P, který obsahuje **aspoň jednu konstantu**.

Herbrandovo universum programu P =
množina všech **uzavřených termů** jazyka $\mathcal{L}_P$, označujeme U_P .

Herbrandova báze programu P =
množina všech **uzavřených atomických formulí** jazyka $\mathcal{L}_P$, označujeme B_P .

Poznámky

- kdyby $\mathcal{L}_P$ neobsahoval ani jednu konstantu, pak bychom měli $U_P = \emptyset$
- pokud P neobsahuje konstantu, pak „nějakou přidáme“
- každý definitní program obsahuje aspoň jeden relační symbol, tedy za předpokladu, že existuje aspoň jedna konstanta, máme $B_P \neq \emptyset$.

Příklad: pro $P = \{even(zero), even(succ(succ(X))) \leftarrow even(X)\}$ máme
$\mathcal{L}_P$ obsahuje funktory $zero/0$ a $succ/1$, a relační symbol $even/1$
$U_P = \{zero, succ(zero), succ(succ(zero)), succ(succ(succ(zero))), \ldots\}$
$B_P = \{even(zero), even(succ(zero)), even(succ(succ(zero))), \ldots\}$

Herbrandova struktura

Pro definitní program P zavedeme jeho *interpretaci.*

- „náš program“ P (který jsme napsali) má námi **zamýšlenou interpretaci**
- chceme umožnit, aby P mohl mít jakoukoliv „rozumnou interpretaci“, protože počítač (interpret PROLOGu) naši zamýšlenou interpretaci nezná
- analogická motivace jako u pojmů **teorie** a **struktura** v predikátové logice

Definice (herbrandovská struktura).
Nechť P je definitní program. Každou podmnožinu $M \subseteq B_P$ nazveme **herbrandovská struktura pro program P**.

Slovně: herbrandovská struktura = podmnožina herbrandovské báze.

Příklad: pro $P = \{even(zero), even(s(s(X))) \leftarrow even(X)\}$ máme

$$B_P = \{even(zero), even(s(zero)), even(s(s(zero))), \dots\}$$
$$M_1 = \{even(zero)\}$$
$$M_2 = \{even(zero), even(s(s(s(s(zero))))))\}$$
$$M_3 = \{even(zero), even(s(s(zero))), even(s(s(s(s(zero))))), \dots\}$$
$$M_4 = \{even(s(zero)), even(s(s(s(zero)))), even(s(s(s(s(s(zero)))))), \dots\}$$
$$M_5 = B_P$$
$$\dots$$

Herbrandova struktura a pravdivost

Herbrandovská struktura $M \subseteq B_P$ vymezuje, které *uzavřené atomické formule* z B_P považujeme (v této struktuře) za *pravdivé*. Formálně:

Definice (pravdivost).
Definitní klauzule $A_0 \leftarrow A_1, \ldots, A_n$ je **pravdivá v herbrandovské struktuře M pro definitní program P**, což značíme $M \models A_0 \leftarrow A_1, \ldots, A_n$, pokud pro každou substituci θ takovou, že $\{A_0\theta, \ldots, A_n\theta\} \subseteq B_P$ platí:

$$\text{pokud } \{A_1\theta, \ldots, A_n\theta\} \subseteq M, \quad \text{pak } A_0\theta \in M \ .$$

Speciálně dostáváme (pro $A_0 \leftarrow A_1, \ldots, A_n$ a M stejného jazyka):

- je-li $A_0 \leftarrow A_1, \ldots, A_n$ tvaru $A_0 \leftarrow$ (**fakt**), pak
  $M \models A_0$ p. k. pro každou θ, kde $A_0\theta \in B_P$, platí $A_0\theta \in M$.
- je-li $A_0 \leftarrow A_1, \ldots, A_n$ **uzavřená klauzule** (neobsahuje proměnné), pak
  $M \models A_0 \leftarrow A_1, \ldots, A_n$ p. k. platí: pokud $\{A_1, \ldots, A_n\} \subseteq M$ pak $A_0 \in M$.
- je-li $A_0 \leftarrow A_1, \ldots, A_n$ uzavřená klauzule tvaru $A_0 \leftarrow$ (**uzavřený fakt**), pak
  $M \models A_0$ p. k. $A_0 \in M$. To jest $M = \{A \in B_P \mid M \models A\}$.

Herbrandův model

Definice (herbrandův model).
Herbrandovská struktura M pro definitní program P se nazývá **herbrandův model definitního programu P**, pokud pro každou klauzuli $A_0 \leftarrow A_1, \ldots, A_n$ z programu P máme $M \models A_0 \leftarrow A_1, \ldots, A_n$.

Slovně: M je model P, pokud je každá klauzule z P pravdivá v M.

Triviální pozorování

$M = B_P$ je modelem P, tedy **každý definitní program má model**. (!)

Označení: $\mathrm{Mod}(P)$ = množina všech herbrandovských modelů P.

Příklad: pro $P = \{even(zero), even(s(s(X))) \leftarrow even(X)\}$ máme

$B_P = \{even(zero), even(s(zero)), even(s(s(zero))), \ldots\}$

$M_1 = \{even(zero)\} \notin \mathrm{Mod}(P)$

$M_2 = \{even(zero), even(s(s(s(s(s(zero)))))))\} \notin \mathrm{Mod}(P)$

$M_3 = \{even(zero), even(s(s(zero))), even(s(s(s(s(zero))))), \ldots\} \in \mathrm{Mod}(P)$

$M_4 = M_3 \cup \{even(s^{2n+1}(z)) \mid n \geq 100\} \in \mathrm{Mod}(P)$

$M_5 = \{even(s(zero)), even(s(s(s(zero)))), \ldots\} \notin \mathrm{Mod}(P)$

$M_6 = B_P \in \mathrm{Mod}(P)$

$\ldots$

Sémantické vyplývání z definitního programu

Formalizujeme sémantický důsledek (vyplývání) z definitního programu.

Definice (sémantické vyplývání).
Nechť P je definitní program a $N \subseteq B_P$. Pokud pro každý $M \in \mathrm{Mod}(P)$ platí $N \subseteq M$, pak **N sémanticky plyne z P**, což značíme $P \models N$. Speciálně pokud $N = \{A_1, \ldots, A_n\}$, pak píšeme $P \models A_1, \ldots, A_n$.

Význam sémantického vyplývání

- definice vyplývání, která nezávisí na našem **zamýšleném modelu**
- uzavřená atomická formule $A \in B_P$ plyne z P, právě když se nachází v každém modelu P, tedy včetně našeho zamýšleného modelu, ať je jakýkoliv

Příklad: pro $P = \{even(zero), even(s(s(X))) \leftarrow even(X)\}$ máme

$P \models even(zero)$

$P \models even(s(s(zero))), even(s(s(s(s(zero)))))$

$P \models \{even(zero), even(s(s(zero))), even(s(s(s(s(zero))))), \ldots\}$

$P \not\models \{even(s(zero))\}$

$P \not\models \{even(s(s(s(zero))))\}$

(protipříklad: $M = \{even(zero), even(s(s(zero))), even(s(s(s(s(zero))))), \ldots\}$)

Odpovědi a korektní odpovědi

Technický problém (motivace pro zavedení „cílů“ a „odpovědí“).
Množina sémantických důsledků definitního programu *může být nekonečná.*

Příklad: pro $P = \{e(z), e(s(s(X))) \leftarrow e(X)\}$ máme
$P \models e(z)$, $P \models e(s(s(z)))$, $P \models e(s(s(s(s(z)))))$, ...

Definice (odpověď).
Nechť P je definitní program a $G = \leftarrow G_1, \ldots, G_m$ je definitní cíl. Substituce θ se nazývá **odpověď pro $P \cup \{G\}$**, pokud $\{G_1\theta, \ldots, G_m\theta\} \subseteq B_P$.

Příklad: pro výše uvedený P a $G = \leftarrow e(s(X))$ máme tyto odpovědi pro $P \cup \{G\}$:
$\theta = \{X/z\}$, $\theta = \{X/s(z)\}$, $\theta = \{X/s(s(z))\}$, ...

Definice (korektní odpověď).
Nechť P je definitní program a $G = \leftarrow G_1, \ldots, G_m$ je definitní cíl. Odpověď θ pro $P \cup \{G\}$ se nazývá **korektní odpověď pro $P \cup \{G\}$**, pokud $P \models G_1\theta, \ldots, G_m\theta$.

Příklad: pro výše uvedený $P \cup \{G\}$ jsou následující odpovědi korektní
$\theta = \{X/s(z)\}$, $\theta = \{X/s(s(s(z)))\}$, $\theta = \{X/s(s(s(s(s(z)))))\}$, ...
následující odpovědi *nejsou* pro $P \cup \{G\}$ korektní:
$\theta = \{X/z\}$, $\theta = \{X/s(s(z))\}$, $\theta = \{X/s(s(s(s(z))))\}$, ...

Nejmenší herbrandův model

Jak nalézt/popsat korektní odpovědi?
Pro jejich popis si (překvapivě) vystačíme s *jediným modelem* P. (!)

Tvrzení. Je-li $\{M_i \mid i \in I\}$ (indexový) systém herbrandovských modelů definitního programu P, pak $\bigcap_{i \in I} M_i$ je herbrandovský model P.

Důkaz. Vezměme definitní klauzuli $A_0 \leftarrow A_1, \ldots, A_n$ programu P a substituci θ takovou, že $(A_0 \leftarrow A_1, \ldots, A_n)\theta \in \mathrm{ground}(P)$. Ukážeme $\bigcap_{i \in I} M_i \models A_0 \leftarrow A_1, \ldots, A_n$. Nechť $\{A_1\theta, \ldots, A_n\theta\} \subseteq \bigcap_{i \in I} M_i$, to jest $\{A_1\theta, \ldots, A_n\theta\} \subseteq M_i$ pro každý $i \in I$. Jelikož je každý M_i ($i \in I$) model P, dostáváme $A_0\theta \in M_i$ pro každý $i \in I$. To jest $A_0\theta \in \bigcap_{i \in I} M_i$. Prokázali jsme tedy, že $\bigcap_{i \in I} M_i \models A_0 \leftarrow A_1, \ldots, A_n$. Jelikož jsme $A_0 \leftarrow A_1, \ldots, A_n$ volili libovolně, dostáváme, že $\bigcap_{i \in I} M_i$ je model P. □

Definice (nejmenší herbrandův model).
Nechť $\{M_i \mid i \in I\}$ je systém všech herbrandových modelů programu P. Herbrandova struktura $M_P \subseteq B_P$ definovaná $M_P = \bigcap_{i \in I} M_i$ se nazývá **nejmenší herbrandův model P**.

Dle předchozího tvrzení:

(i) M_P je model P;
(ii) pro každý herbrandův model M programu P platí $M_P \subseteq M$.

Nejmenší herbrandův model

Příklad 1.

```
male(bill). male(john). male(george). female(monica). female(jane).
isChildOf(john,george). isChildOf(bill,jane). isChildOf(monica,john).
isSonOf(X,Y) :- isChildOf(X,Y), male(X).
isDoughterOf(X,Y) :- isDoughterOf(X,Y), female(X).
isOffspringOf(X,Y) :- isChildOf(X,Y).
isOffspringOf(X,Y) :- isChildOf(X,Z), isOffspringOf(Z,Y).
```

$$M_P = \{male(bill), male(john), male(george), female(monica), female(jane),$$
$$isChildOf(john, george), isChildOf(bill, jane), isChildOf(monica, john),$$
$$isSonOf(bill, jane), isSonOf(john, george), isDoughterOf(monica, john),$$
$$isOffspringOf(john, george), isOffspringOf(bill, jane),$$
$$isOffspringOf(monica, john), isOffspringOf(monica, george)\}.$$

Příklad 2.

```
even(zero).
even(succ(succ(X))) :- even(X).
```

$$M_P = \{even(zero), even(succ(succ(zero))), even(succ(succ(succ(succ(zero))))), \ldots\} =$$
$$= \{even(succ^{2n}(zero)) \mid n \in \mathbb{N}_0\},$$ kde $succ^k$ znamená $succ$ k-krát vnořené

Charakterizace korektních odpovědí

Díky *nejmenšímu herbrandovu modelu* M_P *programu* P získáme nový vhled do pojmu *korektní odpověď pro* $P \cup \{G\}$.

Následující tvrzení říká, že M_P obsahuje **právě všechny důsledky** P, které jsou uzavřenými atomickými formulemi:

Tvrzení. Nechť P je definitní program. Pak $M_P = \{A \in B_P \mid P \models A\}$.

Důkaz. Vezměme $A \in B_P$. Platí, že $P \models A$, právě když pro každý $M \in \mathrm{Mod}(P)$ máme $A \in M$, což je právě když $A \in M_P$. □

Následující tvrzení říká, že korektní odpovědi pro $P \cup \{G\}$ jsou právě ty odpovědi, jejichž aplikací na cíl G získáme prvky M_P:

Tvrzení. Nechť P je definitní program a $G = \leftarrow G_1, \ldots, G_m$ je definitní cíl. Odpověď θ pro $P \cup \{G\}$ je **korektní**, právě když $\{G_1\theta, \ldots, G_m\theta\} \subseteq M_P$.

Důkaz. Dle definice, θ je korektní pro $P \cup \{G\}$ právě když pro každý $M \in \mathrm{Mod}(P)$ máme $\{G_1\theta, \ldots, G_m\theta\} \subseteq M$, což platí právě když $\{G_1\theta, \ldots, G_m\theta\} \subseteq M_P$. □

Shrnutí: Role M_P je klíčová. Otázka je, **jak konstruktivně popsat** M_P**?**

Nalezení nejmenšího herbrandova modelu

Pro definitní program P a libovolnou $I \subseteq B_P$ zavedeme:

$$T_P(I) = \{A_0\theta \,|\, (A_0 \leftarrow A_1, \ldots, A_n)\theta \in \mathrm{ground}(P) \text{ a } \{A_1\theta, \ldots, A_n\theta\} \subseteq I\}.$$

Slovně: $A_0\theta \in T_P(I)$ pokud je $A_0\theta \leftarrow A_1\theta, \ldots, A_n\theta$ uzavřená instance některé definitní klauzule z P a $A_1\theta, \ldots, A_n\theta$ se nachází v I. Intuice: $T_P(I)$ představuje „elementární odvození na *sémantické úrovni*“ z předpokladů I a z programu P.

Pomocí $T_P(I)$ definujeme rekurentně posloupnost $\{T_P^i\}_{i=0}^{\infty}$:

$$T_P^i = \begin{cases} \emptyset & \text{pro } i = 0, \\ T_P^{i-1} \cup T_P(T_P^{i-1}) & \text{pro } i \geq 1. \end{cases}$$

Platí: $T_P^0 \subseteq T_P^1 \subseteq \cdots \subseteq T_P^i \subseteq T_P^{i+1} \subseteq \cdots$
Může však nastat: $T_P^0 \subset T_P^1 \subset \cdots \subset T_P^i \subset T_P^{i+1} \subset \cdots$

Příklad: pro $P = \{e(z), e(s(s(X))) \leftarrow e(X)\}$ máme $T_P^0 = \emptyset$,
$T_P^1 = \{e(z)\}$, $T_P^2 = \{e(z), e(s(s(z))\}$, $T_P^3 = \{e(z), e(s(s(z)), e(s(s(s(s(z))))\}, \ldots$

Prvky posloupnosti $\{T_P^i\}_{i=0}^{\infty}$ sjednotíme a získáme tak T_P^{∞}:

$$T_P^{\infty} = \bigcup_{i=0}^{\infty} T_P^i.$$

Příklad: pro výše uvedený P máme $T_P^{\infty} = \{e(s^{2n}(z)) \,|\, n \in \mathbb{N}_0\}$.

Nalezení nejmenšího herbrandova modelu

Tvrzení. Nechť P je definitní program. Pak $M_P = T_P^\infty$.

Důkaz. „$\subseteq$“: Prokážeme, že T_P^∞ je herbrandův model programu P. Potom ihned dostaneme $M_P \subseteq T_P^\infty$, protože M_P je herbrandův model, který je nejmenší. Dokážeme $T_P^\infty \models A_0 \leftarrow A_1, \ldots, A_n$ pro každou definitní klauzuli $A_0 \leftarrow A_1, \ldots, A_n$ programu P. Uvažujme substituci θ takovou, že $\{A_0\theta, \ldots, A_n\theta\} \subseteq B_P$. Pokud $\{A_1\theta, \ldots, A_n\theta\} \subseteq T_P^\infty$, pak existuje $i \in \mathbb{N}$ takové, že $\{A_1\theta, \ldots, A_n\theta\} \subseteq T_P^i$. To jest $A_0\theta \in T_P(T_P^i) \subseteq T_P^{i+1}$. Odtud máme $T_P^\infty \models A_0 \leftarrow A_1, \ldots, A_n$; T_P^∞ je herbrandův model P.

„$\supseteq$“: Stačí ukázat, že prokaždé $i \in \mathbb{N}_0$ máme $T_P^i \subseteq M_P$. Tvrzení se snadno prokáže indukcí. Pro $i = 0$ je tvrzení triviální. Za předpokladu, že $T_P^i \subseteq M_P$ zřejmě máme, že i $T_P^{i+1} \subseteq M_P$ dle definice $T_P(\cdots)$. □

Pomocí předchozích tvrzení můžeme charakterizovat **korektní odpovědi**:

Tvrzení. Nechť P je definitní program, $G = \leftarrow G_1, \ldots, G_m$ je definitní cíl, a θ je odpověď pro $P \cup \{G\}$. Pak jsou následující tvrzení ekvivalentní:

(i) θ je korektní odpověď pro $P \cup \{G\}$,
(ii) $P \models G_1\theta, \ldots, G_m\theta$,
(iii) $\{G_1\theta, \ldots, G_m\theta\} \subseteq M_P$,
(iv) $\{G_1\theta, \ldots, G_m\theta\} \subseteq T_P^\infty$. □

Deklarativní sémantika definitního programu

korektní odpovědi = odpověď, která (sémanticky) plyne z programu

Postřehy

- definitní programy jsme zatím chápali jako množiny klauzulí
- nezáleží na jejich pořadí, to plně odpovídá deklarativnímu charakteru
- specifikujeme „co“, ale nikoliv „jak“, viz úvodní citát (Lloyd)
- na interpret PROLOGu se budeme (zatím) dívat **zjednodušeně**:
  „PROLOG nám pro **každý dotaz** vrátí **všechny korektní odpovědi**“

Problémy

- charakterizace korektních odpovědí pořád *není konstruktivní*
- definice T_P^∞ je *transfinitní* (obecně nekonečně mnoho kroků),
  při výpočtu T_P^i *neúnosně roste paměť* (exponenciálně s velikostí P)

Řešení problémů

- zavedeme operační (procedurální) sémantiku
- v ideálním případě totožná s deklarativní (třeba v čistém PROLOGu)
- v praxi (překladače PROLOGu) jsou obě sémantiky různé

Operační sémantika definitního programu

Situace: máme zadán definitní program P a definitní cíl G

Úkol (pro překladač PROLOGu): **konstruktivně ověř**, zda-li G plyne z P

- ověření vyplývání, které lze rozdělit do postupných elementárních kroků
- důležitá je konstruktivnost $\Longmapsto$ následná snadná implementace

Postup překladače:

- **z pohledu logického programování** se buduje **posloupnost cílů** a překladač se snaží dospět z výchozího (zadaného) cíle k prázdnému cíli. Výsledná odpověď je stanovena složením substitucí, které se použily během výpočtu.
- **logický pohled**: prokazuje se **spornost teorie** $P \cup \{G\}$; sporností rozumějme spornost definovanou v logickém systému klasické predikátové logiky (viz přednášky ML).

Hledání prázdného cíle (spornosti): **rezoluční metoda** (Robinson, 1965)

- fakty, pravidla a cíl chápeme jako hornovské klauzule
  (tím pádem máme jednotnou „datovou reprezentaci“)
- pomocí rezolučního odvozovacího pravidla se hledá **prázdný cíl**
  (ten odpovídá **nalezení sporu**), podrobněji viz přednášky Mat. logika
- v případě hornovských klauzulí se rezoluční metoda zjednodušuje
  (rezoluce se vždy účastní jeden cíl a jedna programová klauzule)

Motivační příklad

Příklad

```
male(bill). male(john). male(james). female(monica). female(jane).
isChildOf(bill,jane). isChildOf(monica,jane). isChildOf(john,jane).
isChildOf(bill,jane). isSonOf(X,Y) :- isChildOf(X,Y), male(X).
```

Zadán cíl: $\leftarrow isSonOf(X, jane)$

Intuitivní řešení:

K tomu, abychom stanovili odpověď na $\leftarrow isSonOf(X, jane)$ stačí stanovit odpověď na $\leftarrow isChildOf(X, jane), male(X)$, což pro $X = john$ a $X = bill$ dostáváme ihned z databáze faktů;
pro žádnou další hodnotu X to již z databáze faktů neplyne.

Co je třeba vyřešit:

- složení výsledné odpovědi (**skládání substitucí**)
- ztotožnění subcíle s hlavou pravidla (**unifikace**)
- výběr subcíle a výběr programové klauzule při odvozování

Skládání substitucí

Připomeňme pojem: Jsou-li $X_1, \ldots, X_n$ proměnné a $t_1, \ldots, t_n$ termy, kde

(i) $X_i \neq t_i$ $(i = 1, \ldots, n)$, ... *povolíme pouze netriviální substituce*
(ii) $X_i \neq X_j$ $(i \neq j)$, ... *proměnné za které nahrazujeme jsou podvou různé*

pak $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$ nazýváme **substituce**.

Definice (skládání substitucí). Nechť θ a σ jsou substituce ve tvaru $\theta = \{X_1/s_1, \ldots, X_n/s_m\}$, $\sigma = \{Y_1/t_1, \ldots, Y_n/t_n\}$. Pak **složaná substituce $\theta\sigma$** je množina tvaru $\{X_1/(s_1\sigma), \ldots, X_n/(s_m\sigma), Y_1/t_1, \ldots, Y_n/t_n\}$ *ze které vypustíme*:

(a) prvky $X_i/(s_i\sigma)$, pro které $X_i = s_i\sigma$, ... *zaručí platnost podmínky* (i)
(b) prvky Y_j/t_j, pro které $Y_j \in \{X_1, \ldots, X_n\}$. ... *zaručí platnost podmínky* (ii)

Příklad: pro $\theta = \{X/f(Z), Y/W\}$ a $\sigma = \{X/a, Z/a, W/Y\}$,
z množiny $\{X/f(a), Y/Y, X/a, Z/a, W/Y\}$
vyjmeme Y/Y dle bodu (a); vyjmeme X/a dle bodu (b), výsledek:
$\theta\sigma = \{X/f(a), Z/a, W/Y\}$.

Základní vlastnosti složených substitucí (viz literaturu)

- Vztah k aplikaci: $(A\theta)\sigma = A(\theta\sigma)$ pro každé A, θ a σ
- Platí: $\theta(\sigma\tau) = (\theta\sigma)\tau$ (**asociativita**), $\theta\iota = \iota\theta = \theta$ (**neutralita** vůči $\iota = \{\}$)
- Obecně *neplatí*: $\theta\sigma = \sigma\theta$ (skládání substitucí **není komutativní**)

Přejmenování proměnných

Z technických důvodů je někdy nutné **přejmenovat proměnné** v klauzuli tak, aby se žádná z nich **nevyskytovala** mezi danými proměnnými.

Problém:

Pro hornovské klauzule $A_0 \leftarrow A_1, \ldots, A_m$ a $B_0 \leftarrow B_1, \ldots, B_n$ chceme zajistit, aby neměly žádnou společnou proměnnou.

Řešení:

Využijeme substituci tvaru $\{X_1/Y_1, \ldots, X_k/Y_k\}$, kde $X_1, \ldots, X_k$ jsou všechny proměnné vyskytující se v klauzuli $A_0 \leftarrow A_1, \ldots, A_m$ a $Y_1, \ldots, Y_k$ jsou podvou různé proměnné, které se nevyskytují v $B_0 \leftarrow B_1, \ldots, B_n$. Pak $(A_0 \leftarrow A_1, \ldots, A_m)\theta$ je **varianta** klauzule $A_0 \leftarrow A_1, \ldots, A_m$, která nemá s $B_0 \leftarrow B_1, \ldots, B_n$ žádnou společnou proměnnou.

Motivace:

Praktický způsob *oddělení proměnných* v rámci *různých klauzulí programu.*

Unifikace a nejobecnější unifikátor

Definice (unifikátor). Substituce θ se nazývá **unifikátor** množiny atomických formulí $\{\varphi_1, \ldots, \varphi_n\}$, pokud $\varphi_1\theta = \varphi_2\theta = \cdots = \varphi_n\theta$.

Množina T atomických formulí se nazývá **unifikovatelná**, pokud existuje aspoň jeden unifikátor T. Je-li $T = \{\varphi, \psi\}$, říkáme, že φ a ψ jsou/nejsou unifikovatelné.

Substituce θ_1 je **obecnější** než substituce θ_2, právě když existuje substituce σ tak, že $\theta_1\sigma = \theta_2$ (to jest θ_2 vznikne „upřesněním z θ_1“).

Definice (nejobecnější unifikátor). **Nejobecnější unifikátor** množiny T atomických formulí je taková unifikace množiny T, která je obecnější než každá jiná unifikace množiny T.

Poznámky

- nejobecnější unifikace není určena jednoznačně
- (některý) nejobecnější unifikátor množiny $\{\varphi, \psi\}$ značíme $\mathrm{mgu}(\varphi, \psi)$.

Příklady: $p(f(X), Z)$ a $blah(Y, a)$ nejsou unifikovatelné (jiné relační symboly)
$\theta_1 = \{X/a, Y/f(a), Z/a\}$ je unifikátorem $p(f(X), Z)$ a $p(Y, a)$
$\theta_2 = \{Y/f(X), Z/a\}$ je neobecnější unifikátor $p(f(X), Z)$ a $p(Y, a)$,
přitom $\theta_1 = \theta_2\sigma$, kde $\sigma = \{X/a\}$

Rezoluce a odvozování

Definice (elementární krok odvození). Nechť $G = \leftarrow G_1, \ldots, G_m$ je definitní cíl a $C = C_0 \leftarrow C_1, \ldots, C_n$ je definitní klauzule. Definitní cíl **G' vznikne z G a C použitím θ** pokud existuje $i \in \{1, \ldots, m\}$ tak, že

(i) G_i je unifikovatelná s C_0,
(ii) $\theta = \mathrm{mgu}(G_i, C_0)$,
(iii) $G' = \leftarrow G_1\theta, \ldots, G_{i-1}\theta, C_1\theta, \ldots, C_n\theta, G_{i+1}\theta, \ldots, G_m\theta$.

Fakt, že G' vznikne z G a C použitím θ zapisujeme $G \vdash^{C,\theta} G'$.

Elementární krok odvození lze zapsat jako **odvozovací pravidlo**:

$$\frac{\leftarrow G_1, \ldots, G_m\,;\;\; C_0 \leftarrow C_1, \ldots, C_n}{\leftarrow G_1\theta, \ldots, G_{i-1}\theta, C_1\theta, \ldots, C_n\theta, G_{i+1}\theta, \ldots, G_m\theta} \quad \text{pro} \quad \theta = \mathrm{mgu}(G_i, C_0)$$

Příklad: pro následující definitní cíle a klauzule

$G_0 = \leftarrow isSonOf(X, george)$, $C_1 = isSonOf(Y, Z) \leftarrow male(Y), isChildOf(Y, Z)$
$G_1 = \leftarrow male(X), isChildOf(X, george)$, $C_2 = isChildOf(jane, george) \leftarrow$
$G_2 = \leftarrow male(jane)$

máme $G_0 \vdash^{C_1,\theta_1} G_1 \vdash^{C_2,\theta_2} G_2$ pro $\theta_1 = \{Y/X, Z/george\}$ a $\theta_2 = \{X/jane\}$.

SLD-derivace

Definice (SLD-derivace). Nechť P je definitní program a G je definitní cíl. **SLD-derivace z $P \cup \{G\}$** se skládá ze

- (konečné / nekonečné) posloupnosti definitních cílů $G = G_0, G_1, G_2, \ldots$,
- posloupnosti variant programových klauzulí $C_1, C_2, \ldots$, kde pro každé i platí, že žádná z proměnných vyskytujících se v C_i se nevyskytuje v $\{G_0, \ldots, G_{i-1}\}$,
- posloupnosti nejobecnějších unifikátorů $\theta_1, \theta_2, \ldots$,

tak, že každá G_{i+1} vznikne z G_i a C_{i+1} použitím θ_{i+1}.

Symbolicky zapisujeme: $G_0 \vdash_{C_1,\theta_1} G_1 \vdash_{C_2,\theta_2} G_2 \vdash_{C_3,\theta_3} G_3 \cdots$

Poznámky.

- SLD = Selection function + Linear derivation + Definite clauses
- SLD-derivace může být **konečná** i **nekonečná**.
- Pro daný $P \cup \{G\}$ existuje obecně *nekonečně mnoho různých SLD-derivací*.
- V SLD-derivaci se *nepoužívají přímo programové klauzule* (to jest definitní klauzule z P), ale jejich **varianty**. Důvodem přejmenování proměnných je zamezení jejich sdílení mezi programovými klauzulemi a cíli.

Typy SLD-derivací

Rozlišujeme následující tři základní typy SLD-derivací:

SLD-refutace z $P \cup \{G\}$

Konečná posloupnost $G_0 \mid\!\xrightarrow{C_1,\theta_1} G_1 \mid\!\xrightarrow{C_2,\theta_2} G_2 \mid\!\xrightarrow{C_3,\theta_3} \cdots \mid\!\xrightarrow{C_n,\theta_n} G_n = \square$,
jejímž posledním prvkem je **prázdný cíl $\square$**.

Logický pohled: SLD-refutace je důkazem spornosti $P \cup \{G\}$.

Konečná neuspívající SLD-derivace z $P \cup \{G\}$

Konečná posloupnost $G_0 \mid\!\xrightarrow{C_1,\theta_1} G_1 \mid\!\xrightarrow{C_2,\theta_2} G_2 \mid\!\xrightarrow{C_3,\theta_3} \cdots \mid\!\xrightarrow{C_n,\theta_n} G_n \neq \square$,
kterou již **nelze prodloužit** dalším **elementárním odvozením**
(to jest G_n není unifikovatelná s hlavou žádné programové klausule).

Logický pohled: neuspívající SLD-derivace není důkazem spornosti $P \cup \{G\}$,
jedná se o „mrtvou větev výpočtu“, ze které již dál nic nevyvodíme.

Nekonečná SLD-derivace z $P \cup \{G\}$

Posloupnost $G_0 \mid\!\xrightarrow{C_1,\theta_1} G_1 \mid\!\xrightarrow{C_2,\theta_2} G_2 \mid\!\xrightarrow{C_3,\theta_3} G_3 \cdots$ je nekonečná.

Logický pohled: SLD-derivace, která není důkazem spornosti $P \cup \{G\}$,
protože každý definitní cíl G_i je neprázdný.

Vypočtené odpovědi

Abstraktní interpret PROLOGu si lze představit následovně:

Algoritmus (nedeterministický PROLOG).
Vstup: definitní program P a definitní cíl $G = \leftarrow G_1, \ldots, G_m$
Výstup: odpoveď „`No`“, nebo odpověď „`Yes`“ spolu se substitucí θ

> Pokud neexistuje žádná SLD-refutace z $P \cup \{G\}$, pak odpověz „`No`“.
> V opačném případě najdi libovolnou $G_0 \mathrel{|\!\!\frac{C_1,\theta_1}{\quad}} \cdots \mathrel{|\!\!\frac{C_n,\theta_n}{\quad}} G_n = \square$,
> odpověz „`Yes`“ a vrať složenou substituci $\theta = \theta_1\theta_2 \cdots \theta_n$, ze které byly odstraněny všechny X_j/t_j, kde X_j je proměnná nevyskytující se ve G.

Definice (vypočtená odpověď). Nechť P je definitní program a G je definitní cíl. Pokud pro $P \cup \{G\}$ nedeterministický PROLOG vrátí substituci θ, pak se θ nazývá **vypočtená odpověď pro $P \cup \{G\}$**.

Co je nedeterministický algoritmus?

- formální modely nedeterministických algoritmů: viz kurs vyčíslitelnost
- dvě metody *nazírání na nedeterministický počítač*:
  - pokud existuje v daném kroku výpočtu víc možností jak pokračovat, počítač provede „kopii sebe sama“ a každá kopie „jde jiným směrem“
  - algoritmus „uhádne tu správnou výpočetní větev“

Korektní versus vypočtené odpovědi

Tvrzení (korektnost).
Nechť θ je vypočtená odpověď pro $P \cup \{G\}$ a σ je substituce, pro kterou je $G\theta\sigma$ uzavřená instance G. Pak $\theta\sigma$ je korektní odpověď pro $P \cup \{G\}$.

Důkaz. Dokazuje se indukcí přes délku SLD-refutace, přitom vycházíme z ověření korektnosti elementárního odvozovacího kroku, viz literaturu. □

Tvrzení (úplnost).
Je-li θ korektní odpověď pro $P \cup \{G\}$, pak existuje vypočtená odpověď σ pro $P \cup \{G\}$ a substituce τ taková, že $\theta = \sigma\tau$. □

Poznámky

(1) Ad **korektnost**: v předchozím jsme předpokládali, že odpovědi jsou vždy ty substituce, pomocí nichž získáme některou uzavřenou instanci definitního cíle. Jelikož vypočtená odpověď nemusí být de facto odpověď (v tomto smyslu), v tvrzení o korektnost skládáme vypočtenou odpověď s další substitucí σ, která zajistí, že obdržíme uzavřenou instancí cíle.

(2) Ad **úplnost**: tvrzení není přísně vzato „opačné“ k prvnímu – to ani nelze prokázat. Vypočítaná odpověď je vždy „nejobecnější z celé třídy odpovědí“, protože pro její stanovení používáne nejobecnější unifikátory, viz bod (1).

Vypočtená odpověď versus výpočet PROLOGu

Abstraktní algoritmus pro výpočet odpovědi je **nedeterministický** (**!**)

Z pohledu implementace (deterministického) algoritmu musíme specifikovat:

- jak vybírat **subcíl** G_i v cíli $G = \leftarrow G_1, \ldots, G_m$;
- jak vybírat **programovou klauzuli** $C_0 \leftarrow C_1, \ldots, C_n$.
- jak najít **nejobecnější unifikátor** C_0 a G_i, případně jak rozhodnout o tom, zda-li jsou C_0 a G_i unifikovatelné;

Skutečné překladače Prologu vzniknou upřesněním výše popsaného nedeterministického PROLOGu následovně:

- deterministický výběr subcíle G_i: dílčí subcíle v $G = \leftarrow G_1, \ldots, G_m$ chápeme jako *uspořádané zleva doprava*, bere se vždy **první subcíl zleva**;
- deterministický výběru definitní klauzule $C_0 \leftarrow C_1, \ldots, C_n$: definitní klauzule programu (fakty a pravidla) chápeme jako *uspořádané shora dolů* (číslujeme od jedničky), bere se vždy **první možná klauzule shora**;
- k nalezení nejobecnějšího unifikátoru C_0 a G_i se používá **unifikační algoritmus**, který pro C_0 a G_i po konečně mnoha krocích buď vrátí nejobecnější unifikátor nebo odpoví, že formule C_0 a G_i nejsou unifikovatelné

Deterministický výběr subcílů

Výběrová funkce $\mathfrak{R}$ (selection function)

- zavedením výběrové funkce odstraníme nedeterminismus výběru subcíle
- $\mathfrak{R}$ pro každý definitní cíl $\leftarrow G_1, \ldots, G_m$ vrací jeden ze subcílů G_i, píšeme $\mathfrak{R}(\leftarrow G_1, \ldots, G_m) = G_i$.
- máme-li dánu $\mathfrak{R}$, hovoříme o **SLD-derivacích používajících $\mathfrak{R}$**
- otázkou je, jak zavést $\mathfrak{R}$?

Standardní volba $\mathfrak{R}$ = vrať nejlevější subcíl: $\mathfrak{R}(\leftarrow G_1, \ldots, G_m) = G_1$ $(m \geq 1)$

- překladače PROLOGu volí vždy nejlevější subcíl
- analogické s levou derivací při odvozování z bezkontextových gramatik

Lze prokázat tvrzení: pokud je θ vypočtená odpověď pro $P \cup \{G\}$, pak existuje SLD-refutace $G = G_0 \vdash\!\!\!\overset{C_1,\sigma_1}{} \cdots \vdash\!\!\!\overset{C_n,\sigma_n}{} G_n$ z $P \cup \{G\}$ používající $\mathfrak{R}$ tak, že $G\sigma$ je varianta $G\theta$. Z hlediska výpočtu odpovědí **na volbě $\mathfrak{R}$ nezáleží**.

Deterministický výběr programových klauzulí

- netriviální problém výběru programových klauzulí zůstává nevyřešen
- pozor: nelze pouze „zvolit jednu z klauzulí“ (nemuselo by vést k řešení)
- simulujeme nedet. algoritmus: postupně procházíme použitelné klauzule

SLD-stromy

I přesto, že máme dánu výběrovou funkci $\mathfrak{R}$ může nastat situace, že v některém i-tém kroku SLD-derivace $G = G_0 \mid\!\xrightarrow{C_1,\sigma_1} \cdots \mid\!\xrightarrow{C_i,\sigma_i} G_i$ lze použít **různé programové klauzule** C_{i+1} a C'_{i+1} k odvození následujících cílů $G_{i+1} \neq G'_{i+1}$.

– **odvozování není lineární** (ve výše uvedeném smyslu)
– pro zachycení všech možných větví výpočtu zavádíme **SLD-stromy**

Definice (SLD-strom pro $P \cup \{G\}$). SLD-strom je (nekonečný) strom, kde

- **vrcholy** stromu jsou ohodnoceny **definitními cíli**
- **kořen** stromu je G (**počáteční cíl**)
- **hrany** ve stromu jsou **ohodnoceny programovými klauzulemi**
- je-li G' vrchol stromu a pokud $G' \mid\!\xrightarrow{C,\sigma} G''$ (použitím $\mathfrak{R}$), pak **G'' je následník G'** ve stromu a hrana mezi G' a G'' má ohodnocení C.

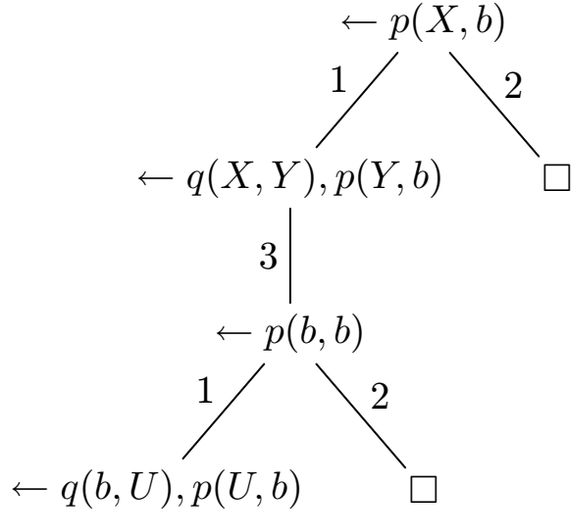
Vztah SLD-stromů a SLD-derivací

- vrcholy SLD-stromu = **stavy výpočtu**
- cesta od kořenu k listu $\square$ = **SLD-refutace**
- cesta od kořenu k listu, který je neprázdný = **neuspívající SLD-derivace**
- cesta od kořenu nekonečnou větví = **nekonečná SLD-derivace**

Příklady SLD-stromů

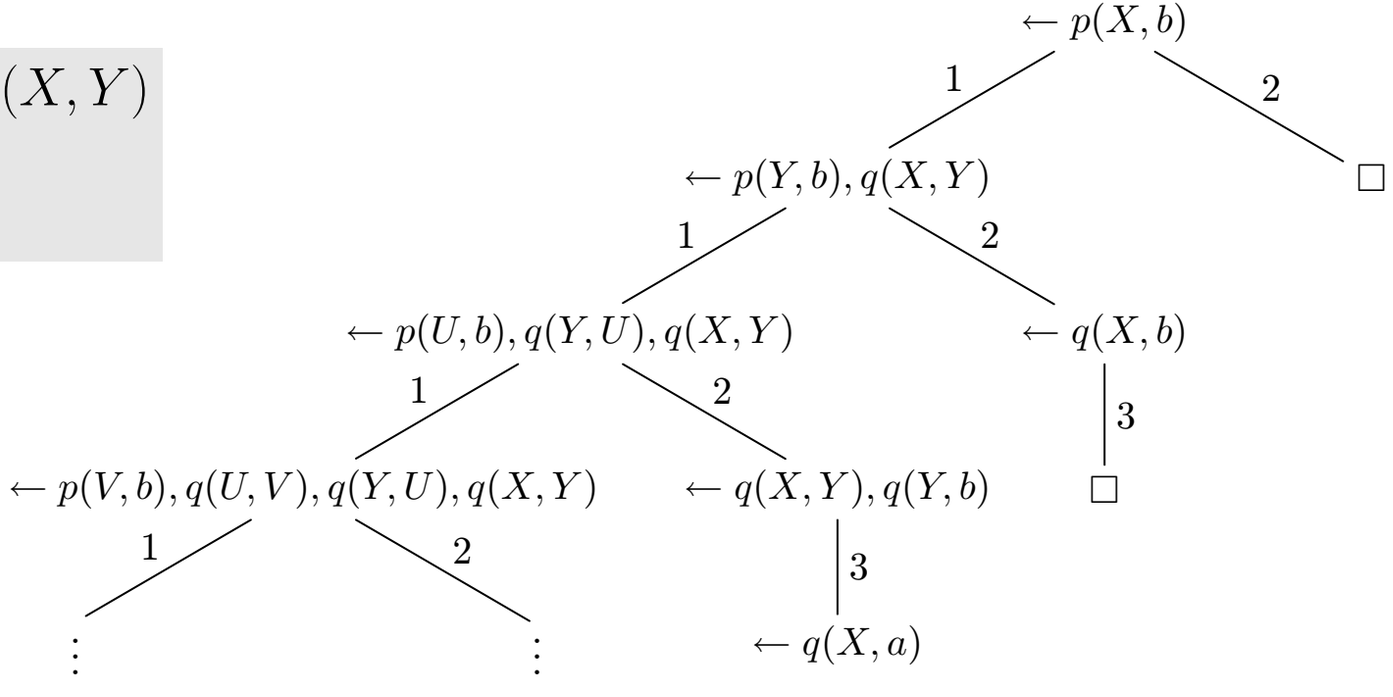
Příklad 1

1. $p(X, Z) \leftarrow q(X, Y), p(Y, Z)$
2. $p(X, X) \leftarrow$
3. $q(a, b) \leftarrow$



Příklad 2

1. $p(X, Z) \leftarrow p(Y, Z), q(X, Y)$
2. $p(X, X) \leftarrow$
3. $q(a, b) \leftarrow$



Strategie prohledávání SLD-stromů

- překladače PROLOGu: systematicky **prohledávají SLD-strom**
- pro jednoznačnost očíslujeme programové klauzule a uzly SLD-stromu

Organizace SLD-stromu

- subcíle v cíli G jsou uspořádané (očíslované zleva doprava)
- programové klauzule jsou uspořádané (očíslované shora dolů)
- pro všechny následovníky $G'_1, \ldots, G'_k$ uzlu G' zavedeme uspořádání $\prec$, že $G'_i \prec G'_j$, právě když hrana $\langle G', G'_i \rangle$ je ohodnocena programovou klauzulí s **ostře menším pořadovým číslem** než hrana $\langle G', G'_j \rangle$

Poznámky

- Dle předchozího: první následovník G' uzlu G, vznikne z $G = \leftarrow G_1, \ldots, G_n$ použitím (přejmenované) programové klauzule $C_i = C_{i,0} \leftarrow C_{i,1}, \ldots, C_{i,n_i}$ s nejmenším číslem (to jest z *první použitelné klauzule bráno shora dolů*), pro kterou je C_0 unifikovatelná s G_1, atd.
- Nyní lze hovořit o prvním, druhém,... následovníku uzlu v SLD-stromě.
- SLD-strom s očíslovaným pořadím vrcholů je možné **prohledávat** běžnými technikami prohledávání stromů (**do hloubky**, **do šířky**).

Strategie prohledávání SLD-stromů

Prohledávání do šířky (breadth first search)

- průchod SLD-stromem po jednotlivých „patrech“ (implementace: **fronta**)
- **výhody:** pokud $\square$ v SLD-stromu existuje, pak ho překladač najde, protože každý vrchol SLD-stromu má nejvýše konečně mnoho následníků
- **nevýhody:** výpočetně náročné (exponenciální paměťová složitost)

Prohledávání do hloubky (depth first search)

- rekurzivní průchod SLD-stromem do hloubky (implementace: **zásobník**)
- **výhody:** výpočetně efektivní (lineární paměťová složitost)
- **nevýhody:** výpočet se může vydat po nekonečné větvi, **vrchol $\square$ nemusí být nalezen** i když se v SLD-stromu nachází
- **překladače PROLOGu** používají **prohledávání do hloubky**

Schéma algoritmu pro průchod do hloubky (implementuje se zásobníkem)

procedura **search**(G):

pokud $G = \square$: vyskoč z rekurze a **skonči úspěchem**

pokud $G \neq \square$ nemá následníky **skonči**

pokud G má následníky $G_1, \ldots, G_m$:

postupně **zavolej** **search**$(G_1), \ldots,$ **search**(G_m)

Základní fáze výpočtu PROLOGu

Inicializace

- **počáteční cíl** = **kořen** procházeného SLD-stromu
- SLD-strom se prochází do hloubky (celý strom se *neudržuje v paměti*)
- podle *směru pohybu ve stromu* dělíme na přímý chod / zpětný chod

Přímý chod

- průchod SLD-stromem **shora dolů** (v programu **zleva doprava**)
- přímý chod představuje jeden *úspěšný elementární krok odvození*
- speciální případ přímého chodu: sestup po **nekonečné větvi**
- sestup po nekonečné větvi obecně **nelze nijak detekovat**

Zpětný chod (**navracení** / **backtracking**)

- průchod SLD-stromem **zdola nahoru** (v programu **zprava doleva**)
- zpětný chod = **návrat k předcházejícímu cíli**
- zpětný chod je vyvolán
  - snahou dostat se ven z neuspívající větve (větev, ve které není řešení)
  - snahou najít alternativní řešení (řešení ve větvi je, ale chceme najít jiné)

Pozor: Programy v PROLOGu je *nutné* psát s ohledem na to, že SLD-strom se prohledává *do hloubky* a *střídají se* předchozí *dvě fáze*; v opačném případě riskujeme uvíznutí výpočtu v „nekonečné větvi“.

Závislost výpočtu PROLOGu na pořadí pravidel

Příklad 1

```
leq(X,Y) :- leq(succ(X),Y).
leq(X,X).
```

Pro dotaz $\leftarrow leq(zero, succ(zero))$ PROLOG začne cyklit:

$\leftarrow leq(zero, succ(zero))$,
$\leftarrow leq(succ(zero), succ(zero))$,
$\leftarrow leq(succ(succ(zero)), succ(zero)), \ldots$

Příklad 2

```
leq(X,X).
leq(X,Y) :- leq(succ(X),Y).
```

Pro dotaz $\leftarrow leq(zero, succ(zero))$ PROLOG vrátí odpověď „`Yes`“.

Příklad 3

```
leq(X,X).
leq(X,succ(Y)) :- leq(X,Y).
```

V tomto případě na pořadí pravidel nezáleží.

Rekursivní datové struktury

Seznamy – plně definovatelné pomocí termů, idea převzatá z LISPu

- **prázdný seznam** značený $[\,]$ je seznam,
- je-li L seznam a a je term, pak term $.(a, L)$ je seznam,
- nic jiného není seznam

Tečkové páry

- roli LISPovského tečkového páru má term $.(a, L)$, kde „$./2$“ je funktor
- místo $.(a, b)$ se používá značení $[a|b]$

Zápis seznamu výčtem

- $[\,]$ . . . prázdný seznam
- $[a]$ . . . jednoprvkový seznam $.(a, [\,])$
- $[a, b, c, d]$ . . . čtyřprvkový seznam $.(a, .(b, .(c, .(d, [\,]))))$
- $[a, [b, c], d]$ . . . tříprvkový seznam $.(a, .(.(b, .(c, [\,])), .(d, [\,])))$

Zápis sezamu pomocí hlavy a těla

- $[H|T]$ . . . alternativní zápis: **hlava H**, **tělo T**
- $[a, b, c, d]$ . . . lze nyní zapsat:

$$[a|[b|[c|[d|[\,]]]]], \qquad [a|[b, c, d]], \qquad [a, b|[c, d]], \qquad [a, b, c|[d]], \qquad [a, b, c, d|[\,]], \ \ldots$$

PROLOG versus Scheme

Zřetězení seznamů (srovněj zápis algoritmu v obou jazycích)

```
(define (append l1 l2)
  (if (null? l1)
      l2
      (cons (car l1)
            (append (cdr l1) l2))))
```

```
append([],L2,L2).
append([X|L1],L2,[X|L]) :- append(L1,L2,L).
```

Převrácení seznamu (srovněj zápis algoritmu v obou jazycích)

```
(define (reverse l)
  (if (null? l)
      '()
      (append (reverse (cdr l))
              (cons (car l) '()))))
```

```
reverse([],[]).
reverse([X|L],R) :- reverse(L,T), append(T,[X],R).
```

Demonstrace významu proměnných

```
append([],List,List).
append([X|List1],List2,[X|List]) :- append(List1,List2,List).
```

Najdi zřetězení seznamů: $\leftarrow append([a,b],[c,d,e],X)$

Vypočtená odpověď: $X = [a,b,c,d,e]$

Najdi prefix seznamu: $\leftarrow append(X,[c,d,e],[a,b,c,d,e])$

Vypočtená odpověď: $X = [a,b]$

Najdi suffix seznamu: $\leftarrow append([a,b],X,[a,b,c,d,e])$

Vypočtená odpověď: $X = [c,d,e]$

Najdi všechna dělení seznamu: $\leftarrow append(X,Y,[a,b,c,d,e])$

Vypočtené odpovědi: $X = [\,],\ Y = [a,b,c,d,e],$

$X = [a],\ Y = [b,c,d,e],$

$X = [a,b],\ Y = [c,d,e],$

$X = [a,b,c],\ Y = [d,e],$

$X = [a,b,c,d],\ Y = [e],$

$X = [a,b,c,d,e],\ Y = [\,]$

Implementace PROLOGu

Ukážeme implementaci plnohodnotného překladače PROLOGu.

Potřebujeme vyřešit

- unifikační algoritmus
- přesně popsat procházení výpočtového stromu do hloubky
  (simulace procházení stromu pomocí **rezolučního zásobníku**)

Ukázková implementace ve Scheme

- autor Vilém Vychodil, `vilem.vychodil@upol.cz`, Palacký University
  `http://vychodil.inf.upol.cz` (spustitelné ve R^5RS Scheme)
- pedagogická implementace PROLOGu ve Scheme (169 řádků)
- tři etapy vývoje:
  - Stage 1: nalezne nejvýš jedno řešení
  - Stage 2: nalezne (všechna) alternativní řešení
  - Stage 3: implementovány řezy a negace
- ke stažení na `http://vychodil.inf.upol.cz/res/devel/scm-prolog/`

Nalezení nejobecnějšího unifikátoru

Algoritmus (nalezení nejobecnějšího unifikátoru).
Vstup: atomické formule φ a ψ
Výstup: substituce θ, která je nejobecnějším unifikátorem φ a ψ,
nebo odpověď „`failed`“, když φ a ψ nejsou unifikovatelné.

polož $\mathcal{E} = \{\langle \varphi, \psi \rangle\}$
cyklus:
vyber libovolný prvek $\langle s, t \rangle \in \mathcal{E}$
pokud $s = f(s_1, \ldots, s_n)$ a $t = f(t_1, \ldots, t_n)$:
nahraď $\langle s, t \rangle$ v $\mathcal{E}$ dvojicemi $\langle s_1, t_1 \rangle, \ldots, \langle s_n, t_n \rangle$ a **opakuj cyklus**
pokud $s = f(s_1, \ldots, s_m)$ a $t = g(t_1, \ldots, t_n)$, kde $f/m \neq g/n$:
ukonči výpočet odpovědí „`failed`“
pokud $s = t$: vyjmi $\langle s, t \rangle$ z $\mathcal{E}$ a **opakuj cyklus**
pokud $t = X$ a s není proměnná:
nahraď $\langle s, X \rangle$ v $\mathcal{E}$ dvojicí $\langle X, s \rangle$ a **opakuj cyklus**
pokud $s = X$, $X \neq t$ a proměnná X má v $\mathcal{E}$ víc jak jeden výskyt:
pokud se X vyskytuje v t (tzv. *occur check*, viz literaturu):
ukonči výpočet odpovědí „`failed`“
jinak:
nahraď ostatní výskyty X v $\mathcal{E}$ termem t a **opakuj cyklus**
pokud pro žádný $\langle s, t \rangle \in \mathcal{E}$ nelze provést žádnou z předchozích akcí:
vrať $\theta = \{X/t \mid \langle X, t \rangle \in \mathcal{E}\}$

Nalezení nejobecnějšího unifikátoru

Příklad 1

Nalezněte nejobecnější unifikátor $f(X, g(Y))$ a $f(g(Z), Z)$:

$\mathcal{E} = \{\langle f(X, g(Y)), f(g(Z), Z)\rangle\},$
$\mathcal{E} = \{\langle X, g(Z)\rangle, \langle g(Y), Z\rangle\},$
$\mathcal{E} = \{\langle X, g(Z)\rangle, \langle Z, g(Y)\rangle\},$
$\mathcal{E} = \{\langle X, g(g(Y))\rangle, \langle Z, g(Y)\rangle\},$
$\theta = \{X/g(g(Y)), Z/g(Y)\}.$

Příklad 2

Nalezněte nejobecnější unifikátor $f(X, g(X), b)$ a $f(a, g(Z), Z)$:

$\mathcal{E} = \{\langle f(X, g(X), b), f(a, g(Z), Z)\rangle\},$
$\mathcal{E} = \{\langle X, a\rangle, \langle g(X), g(Z)\rangle, \langle b, Z\rangle\},$
$\mathcal{E} = \{\langle X, a\rangle, \langle X, Z\rangle, \langle b, Z\rangle\},$
$\mathcal{E} = \{\langle X, a\rangle, \langle X, Z\rangle, \langle Z, b\rangle\},$
$\mathcal{E} = \{\langle X, a\rangle, \langle a, Z\rangle, \langle Z, b\rangle\},$
$\mathcal{E} = \{\langle X, a\rangle, \langle Z, a\rangle, \langle Z, b\rangle\},$
$\mathcal{E} = \{\langle X, a\rangle, \langle Z, a\rangle, \langle a, b\rangle\},$
odpověď: „`failed`“ (výrazy **nejsou unifikovatelné**).

Rezoluční zásobník PROLOGu

Číslování programových klauzulí a kroků výpočtu

- definitní klauzule daného programu jsou očíslovány přirozenými čísly $1, 2, \ldots$,
- cíle, substituce, čísla pravidel a rezoluční zásobník jsou během výpočtu indexovány shodně číslem **kroku výpočtu** i. Na počátku je krok výpočtu nastaven na $i = 0$ a postupně se inkrementuje.

Značení

- **cíle** značíme $G_0, G_1, \ldots$, přitom G_i má tvar $\leftarrow G_{i,1}, G_{i,2}, \ldots, G_{i,n_i}$,
- **varianty programových klauzulí** značíme $C_l = C_{l,0} \leftarrow C_{l,1}, \ldots, C_{l,n_l}$
  (**Pozor:** během výpočtu vždy uvažujeme variantu C_l, která obsahuje pouze proměnné, které jsme *doposud během výpočtu nepoužili*)
- **substituce** značíme $\theta_1, \theta_2, \ldots$,
- **čísla programových klauzulí** (použitých při rezoluci) značíme $R_0, R_1, \ldots$
- **rezoluční zásobníky** značíme $\mathcal{S}_0, \mathcal{S}_1, \ldots$

Obsah rezolučního zásobníku

- obsahem zásobníku jsou trojice $\langle G_{i_j}, R_{i_j}, \theta_{i_j} \rangle$ (cíl, číslo pravidla, substituce)
- vznikne-li $\mathcal{S}_{i+1}$ ze zásobníku $\mathcal{S}_i$ přidáním $\langle G_{i_j}, R_{i_j}, \theta_{i_j} \rangle$ (na konec $\mathcal{S}_i$), budeme tento fakt zkráceně značit $\mathcal{S}_{i+1} = \mathrm{push}(\mathcal{S}_i, \langle G_{i_j}, R_{i_j}, \theta_{i_j} \rangle)$

Činnost rezolučního zásobníku PROLOGu

Algoritmus (nalezení nejvýše jedné odpovědi).
Vstup: definitní program P a definitní cíl G_0 tvaru $\leftarrow G_{0,1}, G_{0,2}, \ldots, G_{0,n_0}$
Výstup: odpoveď „`No`“, nebo odpověď „`Yes`“ spolu s rezolučním zásobníkem $\mathcal{S}$

A. Inicializace výpočtu.

Na počátku máme cíl G_0 tvaru $\leftarrow G_{0,1}, G_{0,2}, \ldots, G_{0,n_0}$.
Dále položme $R_0 = 1$, $\mathcal{S}_0 = \langle\,\rangle$. Pokračujeme krokem „B“ pro $i = 0$:

B. Průběžný i-tý krok výpočtu.

Uvažujeme *aktuální cíl* G_i ve tvaru $\leftarrow G_{i,1}, G_{i,2}, \ldots, G_{i,n_i}$ a *číslo pravidla* R_i, které lze použít jako první. Interpret se snaží unifikovat podcíl $G_{i,1}$ s hlavou některé varianty programové klauzule s číslem větším nebo rovným R_i, přitom začíná klauzulí číslo R_i a postupuje vzestupně. Mohou nastat následující situace.

1. Hlava klauzule C_l ve tvaru $C_{l,0} \leftarrow C_{l,1}, \ldots, C_{l,n_l}$, kde $l \geq R_i$, *je unifikovatelná* s $G_{i,1}$ (**přímý chod algoritmu**). V tomto případě položme

$$
\begin{aligned}
R_{i+1} &= 1,\\
\theta_{i+1} &= \mathrm{mgu}(G_{i,1}, C_{l,0}),\\
G_{i+1} &= C_{l,1}\theta_{i+1}, \ldots, C_{l,n_l}\theta_{i+1}, G_{i,2}\theta_{i+1}, \ldots, G_{i,n_i}\theta_{i+1},\\
\mathcal{S}_{i+1} &= \mathrm{push}(\mathcal{S}_i, \langle G_i, l, \theta_{i+1}\rangle).
\end{aligned}
$$

Speciálně tedy, pokud je C_l faktem, pak $G_{i+1} = G_{i,2}\theta, \ldots, G_{i,n_i}\theta$. Pokud nyní máme $G_{i+1} = \square$, pak výpočet **končí odpovědí „Yes“** spolu s rezolučním zásobníkem $\mathcal{S} = \mathcal{S}_{i+1}$. Pokud $G_{i+1} \neq \square$, pak pokračujeme bodem „B“ v průběžném kroku $i+1$.

2. Hlava žádné klauzule *není unifikovatelná* s $G_{i,1}$ a $\mathcal{S}_i = \langle\,\rangle$ (prázdný zásobník), výpočet **končí odpovědí „No“**.
3. Hlava žádné klauzule *není unifikovatelná* s $G_{i,1}$ a $\mathcal{S}_i \neq \langle\,\rangle$ (zásobník je neprázdný). Nyní nastává **zpětný chod algoritmu** (**backtracking**), to jest návrat k předchozímu cíli, který je na vrcholu zásobníku. Pro rezoluční zásobník ve tvaru $\mathcal{S}_i = \langle\langle G_{i_1}, R_{i_1}, \theta_{i_1}\rangle, \ldots, \langle G_{i_k}, R_{i_k}, \theta_{i_k}\rangle\rangle$ položíme

$$\begin{aligned} G_{i+1} &= G_{i_k}, \\ R_{i+1} &= R_{i_k} + 1, \\ \mathcal{S}_{i+1} &= \langle\langle G_{i_1}, R_{i_1}, \theta_{i_1}\rangle, \ldots, \langle G_{i_{k-1}}, R_{i_{k-1}}, \theta_{i_{k-1}}\rangle\rangle. \end{aligned}$$

Dále pokračujeme bodem „B“ v průběžném kroku $i+1$.

Stanovení odpovědi z výsledného zásobníku $\mathcal{S}$.
Je-li $\mathcal{S} = \langle\langle G_{i_1}, R_{i_1}, \theta_{i_1}\rangle, \ldots, \langle G_{i_k}, R_{i_k}, \theta_{i_k}\rangle\rangle$, pak **výsledná substituce θ** je složená substituce $\theta = \theta_{i_1}\theta_{i_2}\cdots\theta_{i_k}$, ze které odstraníme takové X_j/t_j, kde X_j je proměnná nevyskytující se ve $G_0 = \leftarrow G_{0,1}, G_{0,2}, \ldots, G_{0,n_0}$ (výchozí cíl).

Příklad hledání řešení

Očíslované programové klauzule (zřetězení seznamů)

1. $append(nil, L, L) \leftarrow$
2. $append(cons(X, L), R, cons(X, Y)) \leftarrow append(L, R, Y)$

Zadaný cíl (najdi zřetězení seznamů)

$\leftarrow append(cons(a, cons(b, nil)), cons(c, cons(d, nil)), X)$

Průběh výpočtu

0. $\leftarrow \boldsymbol{append(cons(a, cons(b, nil)), cons(c, cons(d, nil)), X)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_1 = \{X_0/a, L_0/cons(b, nil), R_0/cons(c, cons(d, nil)), X/cons(a, Y_0)\}$

1. $\leftarrow \boldsymbol{append(cons(b, nil), cons(c, cons(d, nil)), Y_0)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_2 = \{X_1/b, L_1/nil, R_1/cons(c, cons(d, nil)), Y_0/cons(b, Y_1)\}$

2. $\leftarrow \boldsymbol{append(nil, cons(c, cons(d, nil)), Y_1)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_3 = \{L_2/cons(c, cons(d, nil)), Y_1/cons(c, cons(d, nil))\}$

3. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_1\theta_2\theta_3 = \{X/cons(a, cons(b, cons(c, cons(d, nil))))\}}$

Příklad hledání řešení

Očíslované programové klauzule (zřetězení seznamů)

1. $append(nil, L, L) \leftarrow$
2. $append(cons(X, L), R, cons(X, Y)) \leftarrow append(L, R, Y)$

Zadaný cíl (najdi prefix seznamu)

$\leftarrow append(X, cons(c, cons(d, nil)), cons(a, cons(b, cons(c, cons(d, nil)))))$

Průběh výpočtu

0. $\leftarrow \boldsymbol{append(X, cons(c, cons(d, nil)), cons(a, cons(b, cons(c, cons(d, nil)))))}$, zač. pr. 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_1 = \{X/cons(a, L_0), R_0/cons(c, cons(d, nil)), X_0/a, Y_0/cons(b, cons(c, cons(d, nil)))\}$

1. $\leftarrow \boldsymbol{append(L_0, cons(c, cons(d, nil)), cons(b, cons(c, cons(d, nil))))}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_2 = \{L_0/cons(b, L_1), R_1/cons(c, cons(d, nil)), X_1/b, Y_1/cons(c, cons(d, nil))\}$

2. $\leftarrow \boldsymbol{append(L_1, cons(c, cons(d, nil)), cons(c, cons(d, nil)))}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_3 = \{L_1/nil, L_2/cons(c, cons(d, nil))\}$

3. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_1\theta_2\theta_3 = \{X/cons(a, cons(b, nil))\}}$

Příklad hledání řešení

Očíslované programové klauzule (zřetězení seznamů)

1. $append(nil, L, L) \leftarrow$
2. $append(cons(X, L), R, cons(X, Y)) \leftarrow append(L, R, Y)$

Zadaný cíl (najdi suffix seznamu)

$\leftarrow append(cons(a, cons(b, nil)), X, cons(a, cons(b, cons(c, cons(d, nil)))))$

Průběh výpočtu

0. $\leftarrow \boldsymbol{append(cons(a, cons(b, nil)), X, cons(a, cons(b, cons(c, cons(d, nil)))))}$, zač. pr. 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_1 = \{X_0/a, L_0/cons(b, nil), X/R_0, Y_0/cons(b, cons(c, cons(d, nil)))\}$

1. $\leftarrow \boldsymbol{append(cons(b, nil), R_0, cons(b, cons(c, cons(d, nil))))}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_2 = \{X_1/b, L_1/nil, R_0/R_1, Y_1/cons(c, cons(d, nil))\}$

2. $\leftarrow \boldsymbol{append(nil, R_1, cons(c, cons(d, nil)))}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_3 = \{R_1/cons(c, cons(d, nil)), L_2/cons(c, cons(d, nil))\}$

3. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_1\theta_2\theta_3 = \{X/cons(c, cons(d, nil))\}}$

Příklad hledání řešení

Očíslované programové klauzule (zřetězení a reverze seznamu)

1. $append(nil, L, L) \leftarrow$
2. $append(cons(X, L), R, cons(X, Y)) \leftarrow append(L, R, Y)$
3. $reverse(nil, nil) \leftarrow$
4. $reverse(cons(X, L), R) \leftarrow reverse(L, T), append(T, cons(X, nil), R)$

Zadaný cíl (převrať seznam)

$\leftarrow reverse(cons(a, cons(b, nil)), X)$

Průběh výpočtu (začátek)

---

0. $\leftarrow \boldsymbol{reverse(cons(a, cons(b, nil)), X)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_1 = \{X_0/a, L_0/cons(b, nil), X/R_0\}$

---

1. $\leftarrow \boldsymbol{reverse(cons(b, nil), T_0), append(T_0, cons(a, nil), R_0)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_2 = \{X_1/b, L_1/nil, T_0/R_1\}$

---

2. $\leftarrow \boldsymbol{reverse(nil, T_1), append(T_1, cons(b, nil), R_1), append(R_1, cons(a, nil), R_0)}$, zač. pr. 1
   subcíl unifikovatelný s hlavou pravidla číslo 3
   $\theta_3 = \{T_1/nil\}$

Příklad hledání řešení

Očíslované programové klauzule (zřetězení a reverze seznamu)

1. $append(nil, L, L) \leftarrow$
2. $append(cons(X, L), R, cons(X, Y)) \leftarrow append(L, R, Y)$
3. $reverse(nil, nil) \leftarrow$
4. $reverse(cons(X, L), R) \leftarrow reverse(L, T), append(T, cons(X, nil), R)$

Průběh výpočtu (zakončení)

---

3. $\leftarrow \boldsymbol{append(nil, cons(b, nil), R_1), append(R_1, cons(a, nil), R_0)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_4 = \{L_3/cons(b, nil), R_1/cons(b, nil)\}$

---

4. $\leftarrow \boldsymbol{append(cons(b, nil), cons(a, nil), R_0)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_5 = \{X_4/b, L_4/nil, R_4/cons(a, nil), R_0/cons(b, Y_4)\}$

---

5. $\leftarrow \boldsymbol{append(nil, cons(a, nil), Y_4)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_6 = \{L_5/cons(a, nil), Y_4/cons(a, nil)\}$

---

6. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_1\theta_2\theta_3\theta_4\theta_5\theta_6 = \{X/cons(b, cons(a, nil))\}}$

Příklad hledání řešení s navracením

Očíslované programové klauzule

1. $male(john) \leftarrow$
2. $male(george) \leftarrow$
3. $male(bill) \leftarrow$
4. $isChildOf(jane, george) \leftarrow$
5. $isChildOf(john, george) \leftarrow$
6. $isChildOf(jane, bill) \leftarrow$
7. $isSonOf(X, Y) \leftarrow isChildOf(X, Y), male(X)$

Průběh výpočtu po zadání cíle $\leftarrow isSonOf(X, george)$

---

0. $\leftarrow \boldsymbol{isSonOf(X, george)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 7
   $\theta_1 = \{X/X_0, Y_0/george\}$

---

1. $\leftarrow \boldsymbol{isChildOf(X_0, george)}, \boldsymbol{male(X_0)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_2 = \{X_0/jane\}$

---

2. $\leftarrow \boldsymbol{male(jane)}$, začínáme pravidlem 1
   subcíl **nelze unifikovat**, návrat k cíli na řádku 1

---

3. $\leftarrow \boldsymbol{isChildOf(X_0, george)}, \boldsymbol{male(X_0)}$, začínáme pravidlem 5
   subcíl unifikovatelný s hlavou pravidla číslo 5
   $\theta_4 = \{X_0/john\}$

Příklad hledání řešení s navracením

Očíslované programové klauzule

1. $male(john) \leftarrow$
2. $male(george) \leftarrow$
3. $male(bill) \leftarrow$
4. $isChildOf(jane, george) \leftarrow$
5. $isChildOf(john, george) \leftarrow$
6. $isChildOf(jane, bill) \leftarrow$
7. $isSonOf(X, Y) \leftarrow isChildOf(X, Y), male(X)$

Průběh výpočtu (zbytek výpočtu)

---

3. $\leftarrow \boldsymbol{isChildOf(X_0, george), male(X_0)}$, začínáme pravidlem 5
   subcíl unifikovatelný s hlavou pravidla číslo 5
   $\theta_4 = \{X_0/john\}$

---

4. $\leftarrow \boldsymbol{male(john)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_5 = \{\}$

---

5. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_1\theta_4\theta_5 = \{X/john\}}$

Poznámky:

- substituce $\theta_2 = \{X_0/jane\}$ se při výpočtu odpovědi neuplatní
- θ_2 při nalezení $\square$ již není na rezolučním zásobníku (odstraněna v kroku 2.)

Příklad hledání řešení s navracením

Očíslované programové klauzule (cesta v acyklickém grafu)

1. $edge(a, b) \leftarrow$
2. $edge(b, d) \leftarrow$
3. $edge(c, d) \leftarrow$

4. $path(X, Y) \leftarrow edge(Z, Y), path(X, Z)$
5. $path(X, Y) \leftarrow edge(X, Y)$

záměrně nevhodné pořadí pravidel (!)

Průběh výpočtu po zadání cíle $\leftarrow path(X, b)$

---

0. $\leftarrow \boldsymbol{path(X, b)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_1 = \{X/X_0, Y_0/b\}$

---

1. $\leftarrow \boldsymbol{edge(Z_0, b), path(X_0, Z_0)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_2 = \{Z_0/a\}$

---

2. $\leftarrow \boldsymbol{path(X_0, a)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_3 = \{X_0/X_2, Y_2/a\}$

---

3. $\leftarrow \boldsymbol{edge(Z_2, a), path(X_2, Z_2)}$, začínáme pravidlem 1
   subcíl **nelze unifikovat**, návrat k cíli na řádku 2

---

4. $\leftarrow \boldsymbol{path(X_0, a)}$, začínáme pravidlem 5
   subcíl unifikovatelný s hlavou pravidla číslo 5
   $\theta_5 = \{X_0/X_4, Y_4/a\}$

Příklad hledání řešení s navracením

Očíslované programové klauzule (cesta v acyklickém grafu)

1. $edge(a, b) \leftarrow$
2. $edge(b, d) \leftarrow$
3. $edge(c, d) \leftarrow$
4. $path(X, Y) \leftarrow edge(Z, Y), path(X, Z)$
5. $path(X, Y) \leftarrow edge(X, Y)$

Průběh výpočtu (dokončení)

5. $\leftarrow \boldsymbol{edge(X_4, a)}$, začínáme pravidlem 1
   subcíl **nelze unifikovat**, návrat k cíli na řádku 2

6. $\leftarrow \boldsymbol{path(X_0, a)}$, začínáme pravidlem 6
   subcíl **nelze unifikovat**, návrat k cíli na řádku 1

7. $\leftarrow \boldsymbol{edge(Z_0, b), path(X_0, Z_0)}$, začínáme pravidlem 2
   subcíl **nelze unifikovat**, návrat k cíli na řádku 0

8. $\leftarrow \boldsymbol{path(X, b)}$, začínáme pravidlem 5
   subcíl unifikovatelný s hlavou pravidla číslo 5
   $\theta_9 = \{X/X_8, Y_8/b\}$

9. $\leftarrow \boldsymbol{edge(X_8, b)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_{10} = \{X_8/a\}$

10. $\square$
    vypočtená odpověď: $\boldsymbol{\theta_9\theta_{10} = \{X/a\}}$

Příklad oznámění neexistence řešení

Očíslované programové klauzule

1. $male(john) \leftarrow$
2. $male(george) \leftarrow$
3. $male(bill) \leftarrow$
4. $isChildOf(jane, george) \leftarrow$
5. $isChildOf(john, george) \leftarrow$
6. $isChildOf(jane, bill) \leftarrow$
7. $isSonOf(X, Y) \leftarrow isChildOf(X, Y), male(X)$

Průběh výpočtu po zadání cíle $\leftarrow isSonOf(jane, Y)$

---

0. $\leftarrow \boldsymbol{isSonOf(jane, Y)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 7
   $\theta_1 = \{X_0/jane, Y/Y_0\}$

---

1. $\leftarrow \boldsymbol{isChildOf(jane, Y_0)}, \boldsymbol{male(jane)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_2 = \{Y_0/george\}$

---

2. $\leftarrow \boldsymbol{male(jane)}$, začínáme pravidlem 1
   subcíl **nelze unifikovat**, návrat k cíli na řádku 1

---

3. $\leftarrow \boldsymbol{isChildOf(jane, Y_0)}, \boldsymbol{male(jane)}$, začínáme pravidlem 5
   subcíl unifikovatelný s hlavou pravidla číslo 6
   $\theta_4 = \{Y_0/bill\}$

Příklad oznámění neexistence řešení

Očíslované programové klauzule

1. $male(john) \leftarrow$
2. $male(george) \leftarrow$
3. $male(bill) \leftarrow$
4. $isChildOf(jane, george) \leftarrow$
5. $isChildOf(john, george) \leftarrow$
6. $isChildOf(jane, bill) \leftarrow$
7. $isSonOf(X, Y) \leftarrow isChildOf(X, Y), male(X)$

Průběh výpočtu (zbytek výpočtu)

---

4. $\leftarrow \boldsymbol{male(jane)}$, začínáme pravidlem 1
   subcíl **nelze unifikovat**, návrat k cíli na řádku 1

---

5. $\leftarrow \boldsymbol{isChildOf(jane, Y_0)}, \boldsymbol{male(jane)}$, začínáme pravidlem 7
   subcíl **nelze unifikovat**, návrat k cíli na řádku 0

---

6. $\leftarrow \boldsymbol{isSonOf(jane, Y)}$, začínáme pravidlem 8
   nelze unifikovat, **rezoluční zásobník je prázdný**, odpověz: **„No“**

Poznámky:

- během výpočtu došlo třikrát k navracení k předchozím cílům
- při posledním navracení jsme se navrátili až k výchozímu cíli
- výpočet končí odpovědí „No“ (žádné řešení neexistuje)

Alternativní řešení

Předchozí algoritmus neuměl nalézt alternativní řešení. Nalezení všech alternativních řešení docílíme následující **modifikací kroku 1. z bodu „B“**:

1. Hlava klauzule C_l ve tvaru $C_{l,0} \leftarrow C_{l,1}, \ldots, C_{l,n_l}$, kde $l \geq R_i$, *je unifikovatelná* s $G_{i,1}$ (**přímý chod algoritmu**). V tomto případě položme

$$\begin{aligned} R_{i+1} &= 1, \\ \theta_{i+1} &= \mathrm{mgu}(G_{i,1}, C_{l,0}), \\ G_{i+1} &= C_{l,1}\theta_{i+1}, \ldots, C_{l,n_l}\theta_{i+1}, G_{i,2}\theta_{i+1}, \ldots, G_{i,n_i}\theta_{i+1}, \\ \mathcal{S}_{i+1} &= \mathrm{push}(\mathcal{S}_i, \langle G_i, l, \theta_{i+1} \rangle). \end{aligned}$$

   Pokud nyní máme $G_{i+1} = \square$, pak vypíšeme **„Yes“** a **odpověď stanovenou ze zásobníku $\mathcal{S}_{i+1}$**; pokud uživatel vloží

   (a) ENTER, pak ukončíme výpočet,

   (b) ;, pak nastav nové hodnoty: $G_{i+1} = G_i$, $R_{i+1} = l + 1$, $\mathcal{S}_{i+1} = \mathcal{S}_i$ a pokračuj bodem „B“ v průběžném kroku $i + 1$.

   Pokud $G_{i+1} \neq \square$, pak pokračujeme bodem „B“ v průběžném kroku $i + 1$.

Poznámka: předchozí algoritmus je „interaktivní“, místo vracení rezolučního zásobníku přímo vypisuje nalezené odpovědi

Příklad hledání alternativních řešení

Očíslované programové klauzule (zřetězení seznamů)

1. $append(nil, L, L) \leftarrow$
2. $append(cons(X, L), R, cons(X, Y)) \leftarrow append(L, R, Y)$

Zadaný cíl (najdi všechna dělení seznamu)

$\leftarrow append(X, Y, cons(a, cons(b, cons(c, nil))))$

Průběh výpočtu

---

0. $\leftarrow \boldsymbol{append(X, Y, cons(a, cons(b, cons(c, nil))))}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_1 = \{X/nil, Y/cons(a, cons(b, cons(c, nil))), L_0/cons(a, cons(b, cons(c, nil)))\}$

---

1. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_1 = \{X/nil, Y/cons(a, cons(b, cons(c, nil)))\}}$
   pokus najít alternativní řešení, návrat k pravidlu na řádku 0

---

2. $\leftarrow \boldsymbol{append(X, Y, cons(a, cons(b, cons(c, nil))))}$, začínáme pravidlem 2
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_3 = \{X/cons(a, L_2), Y/R_2, X_2/a, Y_2/cons(b, cons(c, nil))\}$

---

3. $\leftarrow \boldsymbol{append(L_2, R_2, cons(b, cons(c, nil)))}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_4 = \{L_2/nil, R_2/cons(b, cons(c, nil)), L_3/cons(b, cons(c, nil))\}$

Příklad hledání alternativních řešení

Očíslované programové klauzule (zřetězení seznamů)

1. $append(nil, L, L) \leftarrow$
2. $append(cons(X, L), R, cons(X, Y)) \leftarrow append(L, R, Y)$

Průběh výpočtu (pokračování)

4. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_3\theta_4 = \{X/cons(a, nil), Y/cons(b, cons(c, nil))\}}$
   pokus najít alternativní řešení, návrat k pravidlu na řádku 3

5. $\leftarrow \boldsymbol{append(L_2, R_2, cons(b, cons(c, nil)))}$, začínáme pravidlem 2
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_6 = \{L_2/cons(b, L_5), R_2/R_5, X_5/b, Y_5/cons(c, nil)\}$

6. $\leftarrow \boldsymbol{append(L_5, R_5, cons(c, nil))}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_7 = \{L_5/nil, R_5/cons(c, nil), L_6/cons(c, nil)\}$

7. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_3\theta_6\theta_7 = \{X/cons(a, cons(b, nil)), Y/cons(c, nil)\}}$
   pokus najít alternativní řešení, návrat k pravidlu na řádku 6

8. $\leftarrow \boldsymbol{append(L_5, R_5, cons(c, nil))}$, začínáme pravidlem 2
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_9 = \{L_5/cons(c, L_8), R_5/R_8, X_8/c, Y_8/nil\}$

Příklad hledání alternativních řešení

Očíslované programové klauzule (zřetězení seznamů)

1. $append(nil, L, L) \leftarrow$
2. $append(cons(X, L), R, cons(X, Y)) \leftarrow append(L, R, Y)$

Průběh výpočtu (dokončení výpočtu)

9. $\leftarrow \boldsymbol{append(L_8, R_8, nil)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_{10} = \{L_8/nil, R_8/nil, L_9/nil\}$

10. $\square$
    vypočtená odpověď: $\boldsymbol{\theta_3\theta_6\theta_9\theta_{10} = \{X/cons(a, cons(b, cons(c, nil))), Y/nil\}}$
    pokus najít alternativní řešení, návrat k pravidlu na řádku 9

11. $\leftarrow \boldsymbol{append(L_8, R_8, nil)}$, začínáme pravidlem 2
    subcíl nelze unifikovat, návrat k cíli na řádku 6

12. $\leftarrow \boldsymbol{append(L_5, R_5, cons(c, nil))}$, začínáme pravidlem 3
    subcíl nelze unifikovat, návrat k cíli na řádku 3

13. $\leftarrow \boldsymbol{append(L_2, R_2, cons(b, cons(c, nil)))}$, začínáme pravidlem 3
    subcíl nelze unifikovat, návrat k cíli na řádku 0

14. $\leftarrow \boldsymbol{append(X, Y, cons(a, cons(b, cons(c, nil))))}$, začínáme pravidlem 3
    nelze unifikovat, zásobník je prázdný, odpověz: „`No`“

Příklad hledání alternativních řešení

Očíslované programové klauzule (potomstvo)

1. $isChildOf(jane, george) \leftarrow$
2. $isChildOf(john, george) \leftarrow$
3. $isChildOf(george, bill) \leftarrow$
4. $isOffspringOf(X, Y) \leftarrow isChildOf(X, Y)$
5. $isOffspringOf(X, Y) \leftarrow isChildOf(Z, Y), isOffspringOf(X, Z)$

Průběh výpočtu po zadání cíle: $\leftarrow isOffspringOf(X, bill)$

---

0. $\leftarrow \boldsymbol{isOffspringOf(X, bill)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_1 = \{X/X_0, Y_0/bill\}$

---

1. $\leftarrow \boldsymbol{isChildOf(X_0, bill)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 3
   $\theta_2 = \{X_0/george\}$

---

2. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_1\theta_2 = \{X/george\}}$
   pokus najít alternativní řešení, návrat k pravidlu na řádku 1

---

3. $\leftarrow \boldsymbol{isChildOf(X_0, bill)}$, začínáme pravidlem 4
   subcíl nelze unifikovat, návrat k cíli na řádku 0

---

4. $\leftarrow \boldsymbol{isOffspringOf(X, bill)}$, začínáme pravidlem 5
   subcíl unifikovatelný s hlavou pravidla číslo 5
   $\theta_5 = \{X/X_4, Y_4/bill\}$

Příklad hledání alternativních řešení

Očíslované programové klauzule (potomstvo)

1. $isChildOf(jane, george) \leftarrow$
2. $isChildOf(john, george) \leftarrow$
3. $isChildOf(george, bill) \leftarrow$
4. $isOffspringOf(X, Y) \leftarrow isChildOf(X, Y)$
5. $isOffspringOf(X, Y) \leftarrow isChildOf(Z, Y), isOffspringOf(X, Z)$

Průběh výpočtu (pokračování)

5. $\leftarrow \boldsymbol{isChildOf(Z_4, bill), isOffspringOf(X_4, Z_4)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 3
   $\theta_6 = \{Z_4/george\}$

6. $\leftarrow \boldsymbol{isOffspringOf(X_4, george)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_7 = \{X_4/X_6, Y_6/george\}$

7. $\leftarrow \boldsymbol{isChildOf(X_6, george)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_8 = \{X_6/jane\}$

8. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_5\theta_6\theta_7\theta_8 = \{X/jane\}}$
   pokus najít alternativní řešení, návrat k pravidlu na řádku 7

9. $\leftarrow \boldsymbol{isChildOf(X_6, george)}$, začínáme pravidlem 2
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_{10} = \{X_6/john\}$

Příklad hledání alternativních řešení

Očíslované programové klauzule (potomstvo)

1. $isChildOf(jane, george) \leftarrow$
2. $isChildOf(john, george) \leftarrow$
3. $isChildOf(george, bill) \leftarrow$
4. $isOffspringOf(X, Y) \leftarrow isChildOf(X, Y)$
5. $isOffspringOf(X, Y) \leftarrow isChildOf(Z, Y), isOffspringOf(X, Z)$

Průběh výpočtu (pokračování)

10. $\square$
    vypočtená odpověď: $\boldsymbol{\theta_5\theta_6\theta_7\theta_{10} = \{X/john\}}$
    pokus najít alternativní řešení, návrat k pravidlu na řádku 7

11. $\leftarrow \boldsymbol{isChildOf(X_6, george)}$, začínáme pravidlem 3
    subcíl nelze unifikovat, návrat k cíli na řádku 6

12. $\leftarrow \boldsymbol{isOffspringOf(X_4, george)}$, začínáme pravidlem 5
    subcíl unifikovatelný s hlavou pravidla číslo 5
    $\theta_{13} = \{X_4/X_{12}, Y_{12}/george\}$

13. $\leftarrow \boldsymbol{isChildOf(Z_{12}, george), isOffspringOf(X_{12}, Z_{12})}$, začínáme pravidlem 1
    subcíl unifikovatelný s hlavou pravidla číslo 1
    $\theta_{14} = \{Z_{12}/jane\}$

14. $\leftarrow \boldsymbol{isOffspringOf(X_{12}, jane)}$, začínáme pravidlem 1
    subcíl unifikovatelný s hlavou pravidla číslo 4
    $\theta_{15} = \{X_{12}/X_{14}, Y_{14}/jane\}$

Příklad hledání alternativních řešení

Očíslované programové klauzule (potomstvo)

1. $isChildOf(jane, george) \leftarrow$
2. $isChildOf(john, george) \leftarrow$
3. $isChildOf(george, bill) \leftarrow$
4. $isOffspringOf(X, Y) \leftarrow isChildOf(X, Y)$
5. $isOffspringOf(X, Y) \leftarrow isChildOf(Z, Y), isOffspringOf(X, Z)$

Průběh výpočtu (pokračování)

15. $\leftarrow \boldsymbol{isChildOf(X_{14}, jane)}$, začínáme pravidlem 1
    subcíl nelze unifikovat, návrat k cíli na řádku 14

16. $\leftarrow \boldsymbol{isOffspringOf(X_{12}, jane)}$, začínáme pravidlem 5
    subcíl unifikovatelný s hlavou pravidla číslo 5
    $\theta_{17} = \{X_{12}/X_{16}, Y_{16}/jane\}$

17. $\leftarrow \boldsymbol{isChildOf(Z_{16}, jane)}, \boldsymbol{isOffspringOf(X_{16}, Z_{16})}$, začínáme pravidlem 1
    subcíl nelze unifikovat, návrat k cíli na řádku 14

18. $\leftarrow \boldsymbol{isOffspringOf(X_{12}, jane)}$, začínáme pravidlem 6
    subcíl nelze unifikovat, návrat k cíli na řádku 13

19. $\leftarrow \boldsymbol{isChildOf(Z_{12}, george)}, \boldsymbol{isOffspringOf(X_{12}, Z_{12})}$, začínáme pravidlem 2
    subcíl unifikovatelný s hlavou pravidla číslo 2
    $\theta_{20} = \{Z_{12}/john\}$

20. $\leftarrow \boldsymbol{isOffspringOf(X_{12}, john)}$, začínáme pravidlem 1
    subcíl unifikovatelný s hlavou pravidla číslo 4
    $\theta_{21} = \{X_{12}/X_{20}, Y_{20}/john\}$

Příklad hledání alternativních řešení

Očíslované programové klauzule (potomstvo)

1. $isChildOf(jane, george) \leftarrow$
2. $isChildOf(john, george) \leftarrow$
3. $isChildOf(george, bill) \leftarrow$
4. $isOffspringOf(X, Y) \leftarrow isChildOf(X, Y)$
5. $isOffspringOf(X, Y) \leftarrow isChildOf(Z, Y), isOffspringOf(X, Z)$

Průběh výpočtu (dokončení)

21. $\leftarrow \boldsymbol{isChildOf(X_{20}, john)}$, začínáme pravidlem 1
    subcíl nelze unifikovat, návrat k cíli na řádku 20
22. $\leftarrow \boldsymbol{isOffspringOf(X_{12}, john)}$, začínáme pravidlem 5
    subcíl unifikovatelný s hlavou pravidla číslo 5
23. $\leftarrow \boldsymbol{isChildOf(Z_{22}, john)}, \boldsymbol{isOffspringOf(X_{22}, Z_{22})}$, začínáme pravidlem 1
    subcíl nelze unifikovat, návrat k cíli na řádku 20
24. $\leftarrow \boldsymbol{isOffspringOf(X_{12}, john)}$, začínáme pravidlem 6
    subcíl nelze unifikovat, návrat k cíli na řádku 13
25. $\leftarrow \boldsymbol{isChildOf(Z_{12}, george)}, \boldsymbol{isOffspringOf(X_{12}, Z_{12})}$, začínáme pravidlem 3
    subcíl nelze unifikovat, návrat k cíli na řádku 6
26. $\leftarrow \boldsymbol{isOffspringOf(X_4, george)}$, začínáme pravidlem 6
    subcíl nelze unifikovat, návrat k cíli na řádku 5
27. $\leftarrow \boldsymbol{isChildOf(Z_4, bill)}, \boldsymbol{isOffspringOf(X_4, Z_4)}$, začínáme pravidlem 4
    subcíl nelze unifikovat, návrat k cíli na řádku 0
28. $\leftarrow \boldsymbol{isOffspringOf(X, bill)}$, začínáme pravidlem 6
    nelze unifikovat, zásobník je prázdný, odpověz: „`No`“

Příklad hledání alternativních řešení

Očíslované programové klauzule (sudá přirozená čísla)

1. $even(zero) \leftarrow$
2. $even(succ(succ(X))) \leftarrow even(X)$

Průběh výpočtu po zadání cíle: $\leftarrow even(X)$ (nekonečně mnoho řešení)

---

0. $\leftarrow \boldsymbol{even(X)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_1 = \{X/zero\}$

---

1. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_1 = \{X/zero\}}$
   pokus najít alternativní řešení, návrat k pravidlu na řádku 0

---

2. $\leftarrow \boldsymbol{even(X)}$, začínáme pravidlem 2
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_3 = \{X/succ(succ(X_2))\}$

---

3. $\leftarrow \boldsymbol{even(X_2)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_4 = \{X_2/zero\}$

---

4. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_3\theta_4 = \{X/succ(succ(zero))\}}$
   pokus najít alternativní řešení, návrat k pravidlu na řádku 3
   ...

Řízení výpočtu pomocí řezů

Řez v definitních programech označujeme „!“

- **řez** = prostředek umožňující **kontrolovat běh výpočtu**
- pomocí řezu můžeme během výpočtu „ořezávat SLD-strom“
- čisté logické programování se problematikou řezů *nezabývá* (řezy jsou těsně spjaty s operační sémantikou PROLOGu)

Příklad definitního programu s řezem

$remove(X, nil, nil) \leftarrow$
$remove(X, cons(X, L), R) \leftarrow !, remove(X, L, R)$
$remove(X, cons(Z, L), cons(Z, R)) \leftarrow remove(X, L, R)$

Dva pohledy na řez

- řez je **mimologická anotace** (která není součástí programu)
- řez je speciální **nulární predikát** (který může být součástí programu)

Oba pohledy jsou důležité

- řezy nemají vliv na deklarativní sémantiku definitních programů
- řezy mají vliv na operační sémantiku PROLOGu
- „!“ jako nulární predikát: didaktické / implementační důvody

Princip řezu

Rodičovský cíl řezu (parent goal) = cíl, který během výpočtu PROLOGu **způsobil odvození cíle obsahujícího daný řez**.

- první subcíl v rodičovském cíli je unifikovatelný s pravidlem obsahujícím řez
- rodičovský cíl řezu vyskytujícího se v cíli G je vždy některý (obecně nepřímý) rodičovský uzel G v daném SLD-stromu

Aktivace řezu

- řez je při **prvním průchodu** (přímý chod) **okamžitě splněn**
- při **navracení** (zpětný chod) se řez **aktivuje**
- **aktivace řezu** způsobí postupné odstranění záznamů z vrcholu rezolučního zásobníku až po **rodičovský cíl**, který je rovněž odstraněn (neformálně: aktivace řezu způsobí návrat o jednu úroveň výpočtu výš, to jest alternativní řešení aktuálního dílčího cíle nebudou zkoušena; příslušné větve výpočtového stromu budou odřezány)

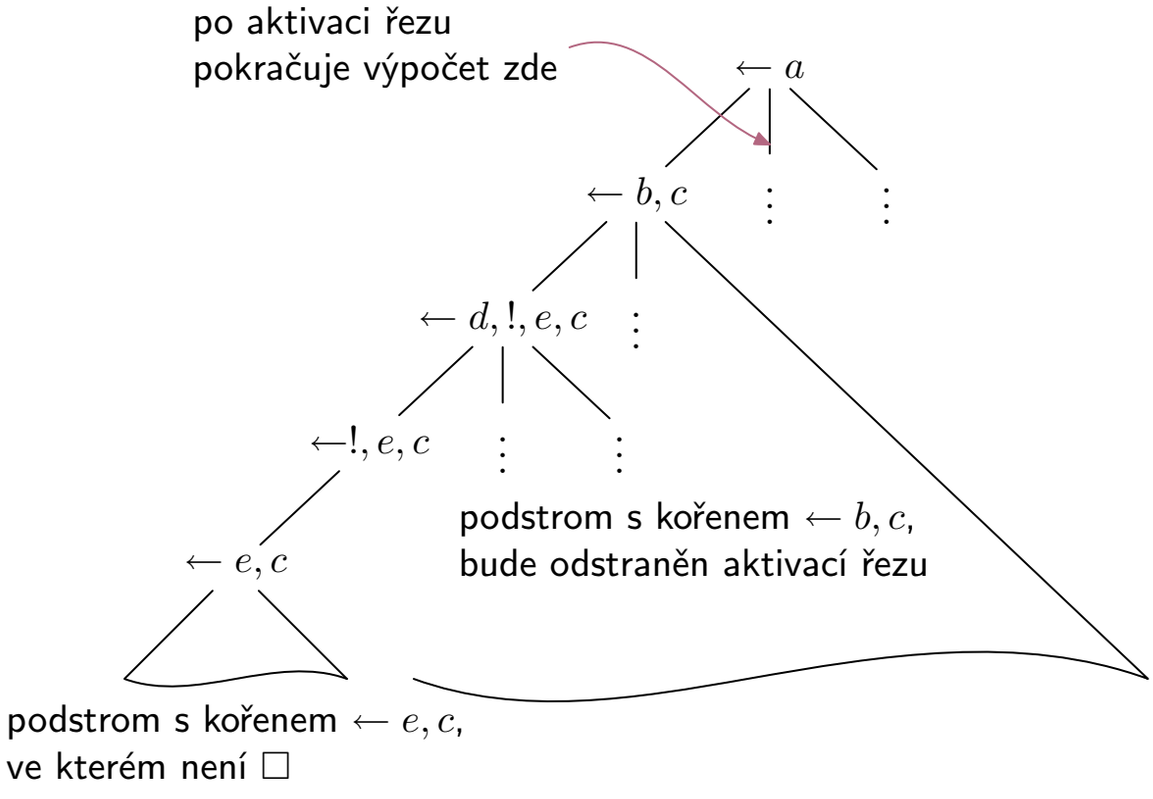
Rozeznáváme dva typy řezů

- **zelený řez** (green/safe cut) – jeho aktivací neztratíme žádné z řešení
- **červený řez** (red/unsafe cut) – řez, který není zelený

Příklad použití řezu

Příklad

$a \leftarrow b, c$
$\cdots$
$b \leftarrow d, \mathbf{!}, e$
$\cdots$
$d \leftarrow$
$\cdots$



Při pokusu o splnění cíle $\leftarrow a$ se bude PROLOG snažit plnit cíl $\leftarrow b, c$ a poté cíl $\leftarrow d, !, e, c$; jelikož je d mezi fakty, dál PROLOG pokračuje plněním cíle $\leftarrow !, e, c$ – jelikož je „!“ v přímém chodu **okamžitě splněn**, PROLOG dál pokračuje plněním cíle $\leftarrow e, c$. Za předpokladu, že $\leftarrow e, c$ neuspěje se PROLOG vrací (ve zpětném chodu) až k cílí $\leftarrow !, e, c$: je **aktivován řez**, který způsobí odstranění celého podstromu s kořenem $\leftarrow b, c$ (rodičovský cíl řezu).

Implementace řezu v PROLOGu

Metoda implementace řezu

- rozšíříme algoritmus PROLOGu (verze pro hledání alternativních řešení)
- zavedeme proměnnou REDO (příznak **směru běhu výpočtu**):
  „REDO = false“ je příznak **přímého chodu**
  „REDO = true“ je příznak **zpětného chodu**
- podle informací v REDO se budeme rozhodovat, zda-li máme řez okamžitě splnit a pokračovat dalším subcílem nebo provést jeho aktivaci a odřezání větví výpočtového stromu

Označení řezů

- řezy budeme reprezentovat jako „speciální unární predikáty“,
  které s sebou mohou nést informaci o rodičovském cíli
- řezy v programových klauzulích značíme „**!**“
- **řezy v cílech** budeme indexovat **čísly kroků výpočtu**: $!_i, !_{i+1}, \ldots$
  (význam: index označuje **číslo rodičovského cíle daného řezu**)
- z technických důvodů zavedeme: $\mathbf{!}\theta = \mathbf{!}$
  (aplikací substituce θ na **!** získáme opět **!**)

Implementace PROLOGu s řezem

Algoritmus (PROLOG s řezem).
Vstup: definitní program P a definitní cíl $G_0 = \leftarrow G_{0,1}, G_{0,2}, \ldots, G_{0,n_0}$ $(n_0 \in \mathbb{N})$
Výstup: bez výstupu (interaktivní algoritmus)

A. Inicializace výpočtu.

Každý výskyt „!“ v G_0 nahradíme $!_0$. Položme $R_0 = 1$, $\mathcal{S}_0 = \langle\,\rangle$, REDO = false.
Dále pokračujeme krokem „B“ pro $i = 0$:

B. Průběžný i-tý krok výpočtu.

Uvažujeme *aktuální cíl* G_i ve tvaru $\leftarrow G_{i,1}, G_{i,2}, \ldots, G_{i,n_i}$ a *číslo pravidla* R_i, které lze použít jako první, zásobník je ve tvaru $\mathcal{S}_i = \langle\langle G_{i_1}, R_{i_1}, \theta_{i_1}\rangle, \ldots, \langle G_{i_k}, R_{i_k}, \theta_{i_k}\rangle\rangle$.
Mohou nastat následující situace.

1. Pokud je $G_i = \square$, pak vypíšeme **„Yes“** a **odpověď stanovenou ze $\mathcal{S}_i$**; pokud uživatel vloží
   (a) ENTER, pak ukončíme výpočet,
   (b) ;, pak nastav hodnoty:

$$G_{i+1} = G_{i_k},$$
$$R_{i+1} = R_{i_k} + 1,$$
$$\mathcal{S}_{i+1} = \langle\langle G_{i_1}, R_{i_1}, \theta_{i_1}\rangle, \ldots, \langle G_{i_{k-1}}, R_{i_{k-1}}, \theta_{i_{k-1}}\rangle\rangle,$$

   REDO = true a pokračuj bodem „B“ v průběžném kroku $i + 1$.

2. Pokud je $G_{i,1} = !_p$, pak provedeme jednu ze dvou alternativ:
   (a) Je-li REDO = false, pak **okamžitě splnímě řez**. Nastavíme

$$G_{i+1} = \leftarrow G_{i,2}, \ldots, G_{i,n_i}$$
$$R_{i+1} = R_i,$$
$$\mathcal{S}_{i+1} = \mathrm{push}(\mathcal{S}_i, \langle G_i, R_i, \iota \rangle). \qquad \iota \ldots \textit{prázdná substituce}$$

   Pokračujeme bodem „B“ v průběžném kroku $i+1$.

   (b) Je-li REDO = true, pak **aktivujeme řez**. Je-li $\mathcal{S}_p = \langle\,\rangle$ (prázdný zásobník), pak výpočet **končí odpovědí „No“**. (aktivací je vyprázdněn zásobník). V opačném případě je $\mathcal{S}_p = \langle \langle G_{i_1}, R_{i_1}, \theta_{i_1} \rangle, \ldots, \langle G_{i_k}, R_{i_h}, \theta_{i_h} \rangle \rangle$. Položíme

$$G_{i+1} = G_{i_h},$$
$$R_{i+1} = R_{i_h} + 1,$$
$$\mathcal{S}_{i+1} = \langle \langle G_{i_1}, R_{i_1}, \theta_{i_1} \rangle, \ldots, \langle G_{i_{h-1}}, R_{i_{h-1}}, \theta_{i_{h-1}} \rangle \rangle.$$

   Pokračujeme bodem „B“ v průběžném kroku $i+1$.

3. Hlava klauzule C_l ve tvaru $C_{l,0} \leftarrow C_{l,1}, \ldots, C_{l,n_l}$, kde $l \geq R_i$, *je unifikovatelná* s $G_{i,1}$ (**přímý chod algoritmu**). V tomto případě položme

$$R_{i+1} = 1,$$
$$\theta_{i+1} = \mathrm{mgu}(G_{i,1}, C_{l,0}),$$
$$G_{i+1} = C_{l,1}\theta_{i+1}, \ldots, C_{l,n_l}\theta_{i+1}, G_{i,2}\theta_{i+1}, \ldots, G_{i,n_i}\theta_{i+1},$$
$$\mathcal{S}_{i+1} = \mathrm{push}(\mathcal{S}_i, \langle G_i, l, \theta_{i+1} \rangle),$$

   přitom **každý výskyt !** v G_{i+1} nahradíme $!_i$.
   Položíme REDO = false a pokračujeme bodem „B“ v kroku $i+1$.

4. Hlava žádné klauzule *není unifikovatelná* s $G_{i,1}$ a $\mathcal{S}_i = \langle\,\rangle$ (prázdný zásobník), výpočet **končí odpovědí „No“**.
5. Hlava žádné klauzule *není unifikovatelná* s $G_{i,1}$ a $\mathcal{S}_i \neq \langle\,\rangle$ (zásobník je neprázdný). Nyní nastává **zpětný chod algoritmu** (**backtracking**), to jest návrat k předchozímu cíli, který je na vrcholu zásobníku. Pro rezoluční zásobník ve tvaru $\mathcal{S}_i = \langle\langle G_{i_1}, R_{i_1}, \theta_{i_1}\rangle, \ldots, \langle G_{i_k}, R_{i_k}, \theta_{i_k}\rangle\rangle$ položíme

$$\begin{aligned} G_{i+1} &= G_{i_k}, \\ R_{i+1} &= R_{i_k} + 1, \\ \mathcal{S}_{i+1} &= \langle\langle G_{i_1}, R_{i_1}, \theta_{i_1}\rangle, \ldots, \langle G_{i_{k-1}}, R_{i_{k-1}}, \theta_{i_{k-1}}\rangle\rangle. \end{aligned}$$

Položíme REDO = true a pokračujeme bodem „B“ v kroku $i+1$.

Příklad červeného řezu

Očíslované programové klauzule (odstranění prvku ze seznamu)

1. $remove(X, nil, nil) \leftarrow$
2. $remove(X, cons(X, L), R) \leftarrow remove(X, L, R), !$
3. $remove(X, cons(Z, L), cons(Z, R)) \leftarrow remove(X, L, R)$

Zadaný cíl $\leftarrow remove(b, cons(b, cons(a, cons(b, nil))), X)$

Průběh výpočtu (zahájení)

---

0. $\leftarrow \boldsymbol{remove(b, cons(b, cons(a, cons(b, nil))), X)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_1 = \{X_0/b, L_0/cons(a, cons(b, nil)), X/R_0\}$

---

1. $\leftarrow \boldsymbol{remove(b, cons(a, cons(b, nil)), R_0)}, \boldsymbol{!_0}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 3
   $\theta_2 = \{X_1/b, Z_1/a, L_1/cons(b, nil), R_0/cons(a, R_1)\}$

---

2. $\leftarrow \boldsymbol{remove(b, cons(b, nil), R_1)}, \boldsymbol{!_0}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_3 = \{X_2/b, L_2/nil, R_1/R_2\}$

---

3. $\leftarrow \boldsymbol{remove(b, nil, R_2)}, \boldsymbol{!_2}, \boldsymbol{!_0}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_4 = \{X_3/b, R_2/nil\}$

Příklad červeného řezu

Očíslované programové klauzule (odstranění prvku ze seznamu)

1. $remove(X, nil, nil) \leftarrow$
2. $remove(X, cons(X, L), R) \leftarrow remove(X, L, R), !$
3. $remove(X, cons(Z, L), cons(Z, R)) \leftarrow remove(X, L, R)$

Zadaný cíl $\leftarrow remove(b, cons(b, cons(a, cons(b, nil))), X)$

Průběh výpočtu (dokončení)

3. $\leftarrow \boldsymbol{remove(b, nil, R_2)}, \boldsymbol{!_2}, \boldsymbol{!_0}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_4 = \{X_3/b, R_2/nil\}$
4. $\leftarrow \boldsymbol{!_2}, \boldsymbol{!_0}$, začínáme pravidlem 1
   přímý chod: řez je okamžitě splněn
5. $\leftarrow \boldsymbol{!_0}$, začínáme pravidlem 1
   přímý chod: řez je okamžitě splněn
6. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_1\theta_2\theta_3\theta_4 = \{X/cons(a, nil)\}}$
   pokus najít alternativní řešení, návrat k pravidlu na řádku 5
7. $\leftarrow \boldsymbol{!_0}$, začínáme pravidlem 2
   zpětný chod: aktivací řezu byl vyprázdněn zásobník, odpověz: „`No`“

Příklad červeného řezu

Očíslované programové klauzule (odstranění prvku ze seznamu)

1. $remove(X, nil, nil) \leftarrow$
2. $remove(X, cons(X, L), R) \leftarrow !, remove(X, L, R)$
3. $remove(X, cons(Z, L), cons(Z, R)) \leftarrow remove(X, L, R)$

změna polohy řezu

Zadaný cíl $\leftarrow remove(b, cons(b, cons(a, cons(b, nil))), X)$

Průběh výpočtu (zahájení)

---

0. $\leftarrow \boldsymbol{remove(b, cons(b, cons(a, cons(b, nil))), X)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_1 = \{X_0/b, L_0/cons(a, cons(b, nil)), X/R_0\}$

---

1. $\leftarrow \boldsymbol{!_0, remove(b, cons(a, cons(b, nil)), R_0)}$, začínáme pravidlem 1
   přímý chod: řez je okamžitě splněn

---

2. $\leftarrow \boldsymbol{remove(b, cons(a, cons(b, nil)), R_0)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 3
   $\theta_3 = \{X_2/b, Z_2/a, L_2/cons(b, nil), R_0/cons(a, R_2)\}$

---

3. $\leftarrow \boldsymbol{remove(b, cons(b, nil), R_2)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_4 = \{X_3/b, L_3/nil, R_2/R_3\}$

---

4. $\leftarrow \boldsymbol{!_3, remove(b, nil, R_3)}$, začínáme pravidlem 1
   přímý chod: řez je okamžitě splněn

Příklad červeného řezu

Očíslované programové klauzule (odstranění prvku ze seznamu)

1. $remove(X, nil, nil) \leftarrow$
2. $remove(X, cons(X, L), R) \leftarrow !, remove(X, L, R)$
3. $remove(X, cons(Z, L), cons(Z, R)) \leftarrow remove(X, L, R)$

změna polohy řezu

Zadaný cíl $\leftarrow remove(b, cons(b, cons(a, cons(b, nil))), X)$

Průběh výpočtu (dokončení)

5. $\leftarrow \boldsymbol{remove(b, nil, R_3)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_6 = \{X_5/b, R_3/nil\}$

6. $\square$
   vypočtená odpověď: $\boldsymbol{\theta_1\theta_3\theta_4\theta_6 = \{X/cons(a, nil)\}}$
   pokus najít alternativní řešení, návrat k pravidlu na řádku 5

7. $\leftarrow \boldsymbol{remove(b, nil, R_3)}$, začínáme pravidlem 2
   subcíl nelze unifikovat, návrat k cíli na řádku 4

8. $\leftarrow \boldsymbol{!_3}, \boldsymbol{remove(b, nil, R_3)}$, začínáme pravidlem 2
   zpětný chod: aktivací řezu jsou odstraněny větve výpočtu až po cíl na řádku 3 včetně

9. $\leftarrow \boldsymbol{remove(b, cons(a, cons(b, nil)), R_0)}$, začínáme pravidlem 4
   subcíl nelze unifikovat, návrat k cíli na řádku 1

10. $\leftarrow \boldsymbol{!_0}, \boldsymbol{remove(b, cons(a, cons(b, nil)), R_0)}$, začínáme pravidlem 2
    zpětný chod: aktivací řezu byl vyprázdněn zásobník, odpověz: „No“

Další příklady červených řezů

Test náležení do seznamu

```
member(X,[X|_]) :- !.
member(X,[_|List]) :- member(X,List).
```

Na rozdíl od bezřezové varianty již X nelze použít jako **výstupní proměnnou** pro nalezení **všech** prvků v seznamu:

```
?- member(X,[a,b,c]). ENTER
X=a ;
No.
```

Nahrazení prvního výskytu elementu

```
replacef([],_,_,[]).
replacef([Y|L],X,Y,[X|L]) :- !.
replacef([A|S],X,Y,[A|L]) :- replacef(S,X,Y,L).
```

Nahrazení posledního výskytu elementu

```
replacel([],_,_,[]).
replacel([A|S],X,Y,[A|L]) :- replacel(S,X,Y,L), member(X,L), !.
replacel([Y|L],X,Y,[X|L]).
```

Příklad zeleného řezu

Očíslované programové klauzule (binární vyhledávací strom)

1. $lt(a, b) \leftarrow$
2. $lt(b, c) \leftarrow$
3. $find(X, node(K, Y, L, Z), S) \leftarrow lt(X, K), !, find(X, L, S)$
4. $find(X, node(K, Y, Z, R), S) \leftarrow lt(K, X), !, find(X, R, S)$
5. $find(X, node(X, V, Y, Z), V) \leftarrow !$
6. $find(X, nil, none) \leftarrow$

Zadaný cíl $\leftarrow find(c, node(b, 20, node(a, 10, nil, nil), node(c, 30, nil, nil)), V)$

Průběh výpočtu (začátek)

---

0. $\leftarrow \boldsymbol{find(c, node(b, 20, node(a, 10, nil, nil), node(c, 30, nil, nil)), V)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 3
   $\theta_1 = \{X_0/c, K_0/b, Y_0/20, L_0/node(a, 10, nil, nil), Z_0/node(c, 30, nil, nil), V/S_0\}$

---

1. $\leftarrow \boldsymbol{lt(c, b)}, \boldsymbol{!_0}, \boldsymbol{find(c, node(a, 10, nil, nil), S_0)}$, začínáme pravidlem 1
   subcíl nelze unifikovat, návrat k cíli na řádku 0

---

2. $\leftarrow \boldsymbol{find(c, node(b, 20, node(a, 10, nil, nil), node(c, 30, nil, nil)), V)}$, začínáme pravidlem 4
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_3 = \{X_2/c, K_2/b, Y_2/20, Z_2/node(a, 10, nil, nil), R_2/node(c, 30, nil, nil), V/S_2\}$

---

3. $\leftarrow \boldsymbol{lt(b, c)}, \boldsymbol{!_0}, \boldsymbol{find(c, node(c, 30, nil, nil), S_2)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_4 = \{\}$

Příklad zeleného řezu

Očíslované programové klauzule (binární vyhledávací strom)

1. $lt(a, b) \leftarrow$
2. $lt(b, c) \leftarrow$

3. $find(X, node(K, Y, L, Z), S) \leftarrow lt(X, K), !, find(X, L, S)$
4. $find(X, node(K, Y, Z, R), S) \leftarrow lt(K, X), !, find(X, R, S)$
5. $find(X, node(X, V, Y, Z), V) \leftarrow !$
6. $find(X, nil, none) \leftarrow$

Průběh výpočtu (průběh)

4. $\leftarrow !_0, \boldsymbol{find(c, node(c, 30, nil, nil), S_2)}$, začínáme pravidlem 1
   přímý chod: řez je okamžitě splněn

5. $\leftarrow \boldsymbol{find(c, node(c, 30, nil, nil), S_2)}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 3
   $\theta_6 = \{X_5/c, K_5/c, Y_5/30, L_5/nil, Z_5/nil, S_2/S_5\}$

6. $\leftarrow \boldsymbol{lt(c, c)}, !_5, \boldsymbol{find(c, nil, S_5)}$, začínáme pravidlem 1
   subcíl nelze unifikovat, návrat k cíli na řádku 5

7. $\leftarrow \boldsymbol{find(c, node(c, 30, nil, nil), S_2)}$, začínáme pravidlem 4
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_8 = \{X_7/c, K_7/c, Y_7/30, Z_7/nil, R_7/nil, S_2/S_7\}$

8. $\leftarrow \boldsymbol{lt(c, c)}, !_5, \boldsymbol{find(c, nil, S_7)}$, začínáme pravidlem 1
   subcíl nelze unifikovat, návrat k cíli na řádku 5

Příklad zeleného řezu

Očíslované programové klauzule (binární vyhledávací strom)

1. $lt(a, b) \leftarrow$
2. $lt(b, c) \leftarrow$

3. $find(X, node(K, Y, L, Z), S) \leftarrow lt(X, K), !, find(X, L, S)$
4. $find(X, node(K, Y, Z, R), S) \leftarrow lt(K, X), !, find(X, R, S)$
5. $find(X, node(X, V, Y, Z), V) \leftarrow !$
6. $find(X, nil, none) \leftarrow$

Průběh výpočtu (dokončení)

9. $\leftarrow \boldsymbol{find(c, node(c, 30, nil, nil), S_2)}$, začínáme pravidlem 5
   subcíl unifikovatelný s hlavou pravidla číslo 5
   $\theta_{10} = \{X_9/c, V_9/30, Y_9/nil, Z_9/nil, S_2/30\}$

10. $\leftarrow \boldsymbol{!_5}$, začínáme pravidlem 1
    přímý chod: řez je okamžitě splněn

11. $\square$
    vypočtená odpověď: $\boldsymbol{\theta_3\theta_{10}\{V/30\}}$
    pokus najít alternativní řešení, návrat k pravidlu na řádku 10

12. $\leftarrow \boldsymbol{!_5}$, začínáme pravidlem 2
    zpětný chod: aktivací řezu jsou odstraněny větve výpočtu až po cíl na řádku 5 včetně

13. $\leftarrow \boldsymbol{!_0, find(c, node(c, 30, nil, nil), S_2)}$, začínáme pravidlem 2
    zpětný chod: aktivací řezu byl vyprázdněn zásobník, odpověz: „`No`“

Další příklady zelených řezů

Databáze popisující ostré uspořádání a rovnost

```
eq(X,X).
lt(a,b). lt(a,c). lt(a,d). lt(a,e). lt(a,f). lt(a,g).
lt(b,c). lt(b,d). lt(b,e). lt(b,f). lt(b,g).
lt(c,d). lt(c,e). lt(c,f). lt(c,g).
lt(d,e). lt(d,f). lt(d,g).
lt(e,f). lt(e,g).
lt(f,g).
```

Vkládání elementu do zatříděného seznamu

```
insert(X,[],[X]) :- !.
insert(X,[Y|List],[Y|S]) :- lt(Y,X), !, insert(X,List,S).
insert(X,[Y|List],[Y|List]) :- eq(X,Y), !.
insert(X,[Y|List],[X,Y|List]) :- lt(X,Y).
```

Třídění postupným zatřiďováním

```
insertsort([],_).
insertsort([X|List],New) :- insertsort(List,S), insert(X,S,New).
```

Další řídící konstrukce

Zabudované predikáty

`fail` ... vždy neuspívá (okamžitě vyvolá navracení)

`true` ... v přímém chodu okamžitě uspěje, při navracení neuspívá

`write(X)`, `nl` ... výpis hodnoty navázané na X, přechod na nový řádek

Zabudované predikáty (cyklus a podmíněný výraz)

```
repeat.
repeat :- repeat.
```

```
if_then_else(Cond,Then,_) :- Cond, !, Then.
if_then_else(_,_,Else) :- Else.
```

Příklad

```
equal(X,X).
nonempty(List) :- equal(List,[_|_]).
find(List) :-
        repeat,
        append(_,[Atom|Tail],List),
        write(Atom), write(' '),
        if_then_else(nonempty(Tail), fail, true),
        !, nl.
```

Negace v PROLOGu

definitní programy = formalizace (naší) **pozitivní znalosti**
Volně řečeno: popisujeme fakty (a další vyplývání z faktů), které *platí*;
nijak nelze zachytit, že nějaký fakt *neplatí.*

Rozšíření definitních programů o negaci (je netriviální)

logický pohled: Z definitního programu (sémanticky/deklarativně) **neplynou negace** žádných **atomických formulí**

Zdůvodnění (užití principu důkazu sporem PL na sémantické úrovni):

Každou definitní klauzuli $A_0 \leftarrow A_1, \ldots, A_n$ reprezentujeme formulí PL:

$(\forall X_1) \cdots (\forall X_k)((A_1 \wedge \cdots \wedge A_n) \rightarrow A_0)$,

kde $X_1, \ldots, X_k$ jsou všechny volné proměnné vyskytující se v $A_0, \ldots, A_n$

Speciálně pro fakt $A_0 \leftarrow$ máme $(\forall X_1) \cdots (\forall X_k) A_0$.

Nyní:

Je-li P definitní program a $A \in B_P$, pak $P \not\models \neg A$, protože $P \cup \{A\}$ má model (B_P je model $P \cup \{A\}$ a tudíž B_P je model P, ve kterém je obsažena A).

Teoretické přístupy k negaci

Pravidlo uzavřeného světa (closed world rule)

– $\neg A$ je důsledkem P, pokud $A \notin M_P$

– slovně: „formule $\neg A$ je důsledkem P, pokud A z P neplyne“

– symbolicky:

$$\frac{P \not\models A}{P \models \neg A}$$

Intuitivně „rozumné“ zavedení (používá se v databázích)

– na úrovni definitních programů je ale algoritmicky neuchopitelné

Negace jako konečné neuspívání (negation as finite failure)

– oslabení pravidla uzavřeného světa

– $\neg A$ je důsledkem P, pokud existuje konečný SLD-strom neobsahující $\square$

– problém: existence výše uvedeného stromu se uvažuje pro nějakou výběrovou funkci $\mathfrak{R}$: pro naši standardní výběrovou funkci „vyber nejlevější subcíl“ může být SLD-strom nekonečný i když některý konečný existuje (pro jinou $\mathfrak{R}$)

– algoritmicky pořád nezvladatelné

Implementace negace v PROLOGu

Zavedení negace v PROLOGu (speciální případ negace jako kon. neusp.)

1. $not(X) \leftarrow X, !, fail$
2. $not(X) \leftarrow$

Pozor: X je proměnná, to jest $not(X) \leftarrow X, !, fail$ přísně vzato není definitní klauzule, protože X je term a nikoliv atomická formule. (!)

X je **metalogická proměnná** (z implementačního pohledu nevadí)

Čisté řešení: zavedeme predikát $call/1$ vyvolající plnění cíle

1. $not(X) \leftarrow call(X), !, fail$
2. $not(X) \leftarrow$

– cíl je navázaný na proměnnou X jako term
– že v PROLOG se mohou funktory a predikáty jménovat stejně

Rovnost: speciální predikát $=/2$

– „vynucení unifikace“

```
X = blah(a,bla), X = Y.     X=blah(a,bla), Y = blah(a,bla)
```

Vlastnosti PROLOGovské negace

Špatná interpretace negace

PROLOGovská negace má **jiné vlastnosti** než **logická negace**

Uvažujme například program

```
isChildOf(john,mary).
isChildOf(jane,bill).
```

Na dotaz

```
?- not(isChildOf(john,X)).
```

PROLOG odpoví „No“,
protože $X = mary$ je odpověď na $\leftarrow isChildOf(john, X)$.

Z pohledu logické negace by však odpovědí (zřejmě) mělo být „Yes“ spolu se substitucí $X = bill$, protože atom $john$ „*není dítětem*“ atomu $bill$.

PROLOGovskou negaci ale nelze chápat jako logickou negaci. (**!**)

Poznámka

Všimněte si, že na X se po splnění cíle $\leftarrow not(isChildOf(john, X))$ nenavázala žádná hodnota (obecný rys zavedení negace pomocí řezu).

Program s negací

Očíslované programové klauzule (modifikované náležení do seznamu)

1. $not(X) \leftarrow X, !, fail$
2. $not(X) \leftarrow$
3. $member(X, cons(X, L)) \leftarrow not(member(X, L))$
4. $member(X, cons(Y, L)) \leftarrow member(X, L)$

Zadaný cíl (zjisti, zda-li je prvek v seznamu)

$\leftarrow member(X, cons(a, cons(a, nil)))$

Průběh výpočtu (zahájení)

---

0. $\leftarrow \boldsymbol{member(X, cons(a, cons(a, nil)))}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 3
   $\theta_1 = \{X/a, X_0/a, L_0/cons(a, nil)\}$

---

1. $\leftarrow \boldsymbol{not(member(a, cons(a, nil)))}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_2 = \{X_1/member(a, cons(a, nil))\}$

---

2. $\leftarrow \boldsymbol{member(a, cons(a, nil))}, \boldsymbol{!_1}, \boldsymbol{fail}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 3
   $\theta_3 = \{X_2/a, L_2/nil\}$

---

3. $\leftarrow \boldsymbol{not(member(a, nil))}, \boldsymbol{!_1}, \boldsymbol{fail}$, začínáme pravidlem 1
   subcíl unifikovatelný s hlavou pravidla číslo 1
   $\theta_4 = \{X_3/member(a, nil)\}$

Program s negací

Očíslované programové klauzule (modifikované náležení do seznamu)

1. $not(X) \leftarrow X, !, fail$
2. $not(X) \leftarrow$
3. $member(X, cons(X, L)) \leftarrow not(member(X, L))$
4. $member(X, cons(Y, L)) \leftarrow member(X, L)$

Průběh výpočtu (využití negace)

4. $\leftarrow \boldsymbol{member(a, nil)}, \boldsymbol{!_3}, \boldsymbol{fail}, \boldsymbol{!_1}, \boldsymbol{fail}$, začínáme pravidlem 1
   subcíl nelze unifikovat, návrat k cíli na řádku 3

5. $\leftarrow \boldsymbol{not(member(a, nil))}, \boldsymbol{!_1}, \boldsymbol{fail}$, začínáme pravidlem 2
   subcíl unifikovatelný s hlavou pravidla číslo 2
   $\theta_6 = \{X_5/member(a, nil)\}$

6. $\leftarrow \boldsymbol{!_1}, \boldsymbol{fail}$, začínáme pravidlem 1
   přímý chod: řez je okamžitě splněn

7. $\leftarrow \boldsymbol{fail}$, začínáme pravidlem 1
   subcíl nelze unifikovat, návrat k cíli na řádku 6

8. $\leftarrow \boldsymbol{!_1}, \boldsymbol{fail}$, začínáme pravidlem 2
   zpětný chod: aktivací řezu jsou odstraněny větve výpočtu až po cíl na řádku 1 včetně

9. $\leftarrow \boldsymbol{member(X, cons(a, cons(a, nil)))}$, začínáme pravidlem 4
   subcíl unifikovatelný s hlavou pravidla číslo 4
   $\theta_{10} = \{X/X_9, Y_9/a, L_9/cons(a, nil)\}$

10. $\leftarrow \boldsymbol{member(X_9, cons(a, nil))}$, začínáme pravidlem 1
    subcíl unifikovatelný s hlavou pravidla číslo 3
    $\theta_{11} = \{X_9/a, X_{10}/a, L_{10}/nil\}$

Program s negací

Očíslované programové klauzule (modifikované náležení do seznamu)

1. $not(X) \leftarrow X, !, fail$
2. $not(X) \leftarrow$
3. $member(X, cons(X, L)) \leftarrow not(member(X, L))$
4. $member(X, cons(Y, L)) \leftarrow member(X, L)$

Průběh výpočtu (dokončení)

11. $\leftarrow \boldsymbol{not(member(a, nil))}$, začínáme pravidlem 1
    subcíl unifikovatelný s hlavou pravidla číslo 1
    $\theta_{12} = \{X_{11}/member(a, nil)\}$
12. $\leftarrow \boldsymbol{member(a, nil)}, \boldsymbol{!_{11}}, \boldsymbol{fail}$, začínáme pravidlem 1
    subcíl nelze unifikovat, návrat k cíli na řádku 11
13. $\leftarrow \boldsymbol{not(member(a, nil))}$, začínáme pravidlem 2
    subcíl unifikovatelný s hlavou pravidla číslo 2
    $\theta_{14} = \{X_{13}/member(a, nil)\}$
14. $\square$
    vypočtená odpověď: $\boldsymbol{\theta_{10}\theta_{11}\theta_{14} = \{X/a\}}$
    pokus najít alternativní řešení, návrat k pravidlu na řádku 11
15. $\leftarrow \boldsymbol{not(member(a, nil))}$, začínáme pravidlem 3: nelze unifikovat, návrat k 10
16. $\leftarrow \boldsymbol{member(X_9, cons(a, nil))}$, začínáme pravidlem 4
    subcíl unifikovatelný s hlavou pravidla číslo 4
17. $\leftarrow \boldsymbol{member(X_{16}, nil)}$, začínáme pravidlem 1: nelze unifikovat, návrat k 10
18. $\leftarrow \boldsymbol{member(X_9, cons(a, nil))}$, začínáme pravidlem 5: nelze unifikovat, návrat k 0
19. $\leftarrow \boldsymbol{member(X, cons(a, cons(a, nil)))}$, začínáme pravidlem 5: odpověz: „`No`“

Další příklady programů s negací

Poslední prvek seznamu

```
empty([]).
last(L,[L]).
last(L,[_|R]) :- not(empty(R)), last(L,R).
```

```
last(L,[L]).
last(L,[_|R]) :- not(R = []), last(L,R).
```

Odstraň dané prvky ze seznamu

```
equals(X,X).
remove(_,[],[]).
remove(X,[X|Y],R) :- remove(X,Y,R).
remove(X,[Q|Y],[Q|R]) :- not(equals(X,Q)), remove(X,Y,R).
```

```
remove(_,[],[]).
remove(X,[X|Y],R) :- remove(X,Y,R).
remove(X,[Q|Y],[Q|R]) :- not(X = Q), remove(X,Y,R).
```

Aritmetika zabudovaná v PROLOGu

Binární predikát „is“

- $X\ is\ Expr$ uspěje pokud je X unifikovatelné s číslem vzniklým **vyhodnocením aritmetického výrazu** $Expr$
- **koerce PROLOGu**: pokud je výsledek vyhodnocení $Expr$ převeditelný na celé číslo, PROLOG provede **přetypování** ($1.0 \Longmapsto 1$, etc.)
- v čistém PROLOGu is nezavádíme – **aritmetika je definovatelná**

Další predikáty

$</2$ (ostře menší), $=</2$ (menší rovno), $>/2$ (ostře větší), $>=/2$ (větší rovno), $=:=/2$ (rovno), $=\backslash=/2$ (nerovno)

Příklady odpovědí na dotazy

`Y = 20, X = Y+1.`	`Y=20, X = 20+1` (na X je navázaný term „20+1“)
`Y = 20, X is Y+1.`	`Y=20, X = 21` (na X navázané číslo)
`Y is 20, X is Y+1.`	`Y=20, X = 21`
`2 = 1+1.`	No (termy „2“ a „1+1“ nejsou unifikovatelné)
`2 is 1+1.`	Yes
`2 =:= 1+1.`	Yes
`X = 10, Y = 20, X is Y+1.`	No
`Y = 20, Y is X+1.`	Error: X is not sufficiently instantiated
`Y = 20, X is Y-1.`	`X = 19`
`1.0 is sin(pi/2).`	No (koerce: `sin(pi/2)` je celé číslo)

Ukázky dalších programů v PROLOGu

Faktoriál

```
factorial(0,1).
factorial(N,Factorial) :-
        N >= 1,
        M is N-1,
        factorial(M,R),
        Factorial is R*N.
```

Seznam přirozených čísel

```
range(A,A,[A]).
range(A,B,[A|X]) :- A < B, C is A+1, range(C,B,X).
```

Variace s opakováním

```
vars(0,_,[]).
vars(K,Set,[X|R]) :-
        K > 0,
        K1 is K-1,
        vars(K1,Set,R),
        member(X,Set).
```

Ukázky dalších programů v PROLOGu

Slití setříděných seznamů

```
merge([],X,X).
merge(X,[],X).
merge([X|L1],[Y|L2],[X|R]) :- X < Y, merge(L1,[Y|L2],R).
merge([X|L1],[Y|L2],[Y|R]) :- X > Y, merge([X|L1],L2,R).
merge([X|L1],[X|L2],[X|R]) :- merge(L1,L2,R).
```

Počet atomů seznamu

```
atoms([],0) :- !.
atoms([H|S],N) :- !,
        atoms(S,N1), atoms(H,N2),
        N is N1+N2.
atoms(_,1).
```

Linearizace seznamu

```
lin([],[]) :- !.
lin([H|S],L) :- !,
        lin(S,L1), lin(H,L2),
        append(L1,L2,L).
lin(X,[X]).
```

Ukázky dalších programů v PROLOGu

Hledání všech cest v grafu

```
pred(a,b). pred(b,c). pred(c,g).
pred(a,d). pred(d,e). pred(e,c).

remove(_,[],[]).
remove(X,[X|Y],R) :- remove(X,Y,R).
remove(X,[Q|Y],[Q|R]) :- not(X = Q), remove(X,Y,R).

edge(X,Y) :- not(X = Y), pred(X,Y).
edge(X,Y) :- not(X = Y), pred(Y,X).

path(X,_,[X]).
path(X,UseSet,[X|Path]) :-
        member(Y,UseSet),
        edge(X,Y),
        remove(X,UseSet,Rest1),
        remove(Y,Rest1,Rest2),
        path(Y,Rest2,Path).
```