
KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI
www.inf.upol.cz

Jan Laštovička
https://www.inf.upol.cz/lide/jan-lastovicka
jan.lastovicka@upol.cz
17. listopadu 12, 771 46 Olomouc

Paradigmata programování 4 ◇ poznámky k přednášce

10. Proměnné v pravidlech

verze z 15. dubna 2025

1 Proměnné

Začněme motivačním příkladem, kde máme zadaná fakta:

1. Abel je dítětem Evy.
2. Abel je dítětem Adama.
3. Adam je muž.
4. Abel je muž.
5. Eva je žena.

a jediné pravidlo:

- Jestliže ***y*** je dítětem ***x*** a ***x*** je muž, pak ***x*** je otcem ***y***.

Ptáme se, zda z faktů a pravidla plyne, že je Adam otec Abela.

V logickém programu budeme modelovat jevy, které dělíme na samostatné a průvodní. Samostatné jevy dokážeme pojmout nezávisle na ostatních jevech. Například Abel je samostatný jev, ale býti mužem už samostatný jev není. Samostatné jevy nazýváme objekty. Průvodní jevy je možné pozorovat jen pomocí objektů. Například jev býti mužem je průvodní a můžeme jej pozorovat například na objektu Adam.

Objekt popíšeme necyklickou strukturou tečkových párů. Zatím si vystačíme se symboly:

- Adam ... `adam`
- Eva ... `eva`
- Abel ... `abel`

Průvodní jevy modelujeme pomocí **predikátů**. Predikát je symbol. Například:

1. býti dítětem ... `child`

2. býti otcem ... `father`
3. býti mužem ... `male`
4. býti ženou ... `female`

Atomickou výrokovou formu zapíšeme seznamem (*predicate arg1 arg2 ...*), kde:

- *predicate* je predikát
- *argi* jsou objekty zvané **argumenty**

Například:

1. Abel je dítětem Evy ... `(child abel eva)`
2. Eva je otcem Abela ... `(father eva abel)`
3. Adam je muž ... `(male adam)`
4. Abel je žena ... `(female abel)`

Proměnná je symbol začínající otazníkem, za kterým následuje aspoň jeden další znak. Například: `?x`, `?person`. Atomické výrokové formy mohou obsahovat proměnné. Například:

- `?y` je dítětem `?x` ... `(child ?y ?x)`
- `?x` je muž ... `(male ?x)`
- `?x` je otcem `?y` ... `(father ?x ?y)`

Atomický výrok je atomická výroková forma bez proměnných. Například: `(child abel eva)`. Atomy lispu, které nejsou proměnné, nazýváme **konstanty**.

Pravidlo je výrok tvaru:

- Pro každé *var1*, ..., *varm* platí, že jestliže *body1* a ... a *bodyn*, pak *head*,

kde

- *bodyi* a *head* jsou atomické výrokové formy
- a *vari* jsou všechny proměnné, které se v nich vyskytují.

Opět zapisujeme: (<- *head body1 ... bodyn*). Zavádíme omezení, že proměnné v těle pravidla se musí vyskytovat i v hlavě pravidla. Například:

- Pro každé `?x` a `?y` platí, že jestliže `?y` je dítětem `?x` a `?x` je muž, pak `?x` je otcem `?y`.
  ... `(<- (father ?x ?y) (child ?y ?x) (male ?x))`

Cíl má stále tvar:

- *atom1* a ... a *atomn*,

kde *atomi* jsou atomické výroky. Cíl opět zapíšeme seznamem `(?` *atom1* ... *atomn*`)`. Například:

- Adam je otcem Abela. ... `(? (father adam abel))`

Protože cíl je konjunkce atomických výroků, nesmí obsahovat proměnné.

2 Instance výrazů

Substituce je seznam `(sub (`*var1* `.` *val1*`)` ... `(`*varn* `.` *valn*`))`, kde *vari* jsou proměnné a *vali* jsou objekty, splňující následující dvě podmínky.

1. proměnné v *car* párů se neopakují
2. proměnné z *car* párů se nevyskytují v *cdr* žádného páru

Příklady:

1. `(sub)`
2. `(sub (?x . abel))`
3. `(sub (?x . abel) (?y . ?z))`

Substituce `(sub)` se nazývá **prázdná substituce**.

Při zápisu atomických výrokových forem používáme tři druhy výrazů: proměnné (`?x`, `?y`, ...), konstanty (`abel`, `1`, ...) a páry (`(abel . adam)`, `(1 2 3)`, ...).

Aplikace substituce *sub* na výraz *expr*:

1. je-li *expr* konstanta, výsledkem je *expr*,
2. je-li *expr* proměnná, zjistí se, zda není rovna některé proměnné *vari* substituce *sub*. Pokud ano, výsledkem je *vali*, pokud ne, je jím *expr*,
3. je-li *expr* pár `(`*a1* `.` *a2*`)`, výsledkem je pár `(`*b1* `.` *b2*`)`, kde každé *bi* vzniklo rekurzivně aplikací substituce *sub* na *ai*.

Například výsledkem aplikace `(sub (?x . abel))`

- na `(male ?x)` je `(male abel)`,
- na `(father ?x ?y)` je `(father abel ?y)`
- a na `(father ?x ?x)` je `(father abel abel)`.

Substituce *sub* je **unifikátor** výrazů *expr1* a *expr2*, jestliže aplikací *sub* na *expr1* a *expr2* dostaneme týž výraz. Například `(sub (?x . abel) (?y . adam))` je unifikátorem výrazů `(child ?x adam)` a `(child abel ?y)`.

Výraz *expr1* je **instancí** výrazu *expr2*, jestliže existuje substituce *sub* taková, že *expr1* je výsledek aplikace *sub* na *expr2*. Například: `(father adam abel)` je instancí `(father ?x ?y)`. Pokud je výraz *expr1* bez proměnných, pak je instancí *expr2*, právě když existuje jejich unifikátor. V předchozím příkladě je `(sub (?x . adam) (?y . abel))` unifikátorem výrazů.

Níže uvedeným postupem rozhodneme, zda je výraz bez proměnných *expr1* instancí výrazu *expr2*, na který jsme aplikovali substituci *sub*, a v kladném případě vrátíme jejich unifikátor. Pokud chceme rozhodnout, zda je *expr1* instancí *expr2*, můžeme za *sub* vzít prázdnou substituci.

Postup rozhodování, zda je *expr1* **instancí** *expr2* se substitucí *sub*:

1. Vytvoříme nový výraz *expr2'* vzniklý aplikací *sub* na *expr2*.
2. Jestliže *expr1* = *expr2'*, výsledkem je substituce *sub*.
3. Je-li *expr2'* proměnná, výsledkem je substituce *sub* s přidaným párem `(`*expr2'* `.` *expr1*`)`.
4. Jsou-li *expr1* a *expr2'* páry, výsledkem je substituce vzniklá při rozhodování, zda pár *expr1* je instancí páru *expr2'* se *sub*.
5. Jinak zamítáme.

Přidání páru `(`*var* `.` *val*`)` k substituci *sub* probíhá takto:

1. zkontroluje se, zda *var* není rovno některému *vari* ze substituce. Pokud je, je to chyba,
2. zkontroluje se, zda se *var* nevyskytuje v *val*. Pokud se vyskytuje, je to chyba,
3. na všechna *vali* se aplikuje substituce `(sub (`*var* `.` *val*`))`,
4. pár se přidá zepředu k *sub*.

Například přidáním páru `(?x . abel)` k substituci `(sub (?y . ?x))` vznikne substituce `(sub (?x . abel) (?y . abel))`.

Postup rozhodování, zda pár `(`*a1* `.` *a2*`)` je **instancí páru** `(`*b1* `.` *b2*`)` se substitucí *sub*:

1. Rozhodneme, zda *a1* je instancí *b1* se *sub*, pokud ano, obdržíme substituci *sub2*, jinak zamítáme.

2. Rozhodneme, zda *a2* je instancí *b2* se *sub2*, pokud ano, obdržíme substituci *sub3*, jinak zamítáme. Výsledkem je substituce *sub3*.

Například výraz `(a . a)` je instancí výrazu `(?x . ?x)`. Rozhodování vrátí unifikátor `(sub (?x . a))`. Výraz `(a . b)` instancí výrazu `(?x . ?x)` není.

3 Splnění cíle

Program je stále seznam pravidel. Například:

```
((<- (child abel eva))
 (<- (child abel adam))
 (<- (male adam))
 (<- (male abel))
 (<- (female eva))
 (<- (father ?x ?y) (child ?y ?x) (male ?x)))
```

Na cíl `(? atom1 ... atomm)` **lze použít** pravidlo `(<- head body1 ... bodyn)`, pokud *atom1* je instancí *head*. **Použitím** obdržíme cíl:

```
(? body1' ... bodyn' atom2 ... atomm)
```

kde *unifier* je unifikátor *atom1* a *head* a *bodyi'* je výsledek aplikace *unifier* na *bodyi*. Použití pravidla na cíl se nazývá **úsudek**.

Například na cíl

```
(? (father adam abel))
```

lze použít pravidlo

```
(<- (father ?x ?y) (child ?y ?x) (male ?x))
```

S použitím unifikátoru

```
(sub (?x . adam) (?y . abel))
```

obdržíme výsledek

```
(? (child abel adam) (male adam))
```

Dále na cíl

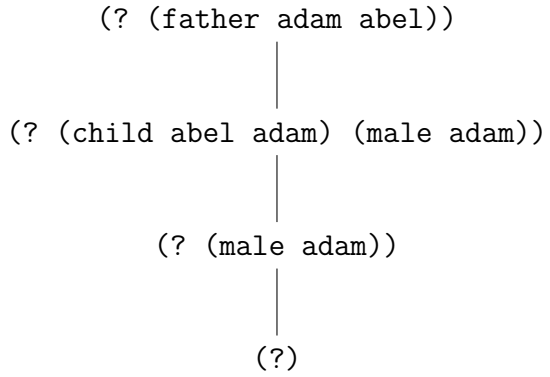
```
(? (child abel adam) (male adam))
```

lze použít pravidlo

```
(<- (child abel adam))
```

Unifikátorem je zde prázdná substituce. Obdržíme výsledek `(? (male adam))`.

Definice strumu úsudků a splnění cíle zůstávají stejné jako v minulé přednášce. Například:



Cíl `(? (father adam abel))` je splněn.

4 Pohled predikátové logiky

Proměnným odpovídají **objektové proměnné**. Objektům odpovídají **termy**. Každá objektová proměnná je termem a aplikace funkčního symbolu na jí zadaný počet termů je termem. Zatím si vystačíme jen s nulárními funkčními symboly, které se také nazývají konstanty. Například:

- `adam` ... *adam*
- `eva` ... *eva*
- `abel` ... *abel*

Predikátům odpovídají **relační symboly**. Například:

- `father` ... $father$

- `male` ... $male$
- `female` ... $female$
- `child` ... $child$

Atomickým výrokovým formám odpovídají **atomické formule**. Například:

- `(father adam abel)` ... $father(adam, abel)$
- `(male ?x)` ... $male(x)$

Pravidlu `(<-` φ ψ_1 ... ψ_n`)` odpovídá formule $(\forall x_1, \ldots, x_m)((\psi_1 \wedge \cdots \wedge \psi_n) \rightarrow \varphi)$, kde $x_1, \ldots, x_m$ jsou všechny proměnné v φ, ψ_1, ..., ψ_n.

Například pravidlu

```
(<- (father ?x ?y) (child ?y ?x) (male ?x))
```

odpovídá formule

$$(\forall x, y)((child(y, x) \wedge male(x)) \rightarrow father(x, y)).$$

Programu odpovídá konečná množina formulí (**teorie**). Teorie pro výše uvedený program:

$$\begin{aligned} T = \{&child(abel, eva), child(abel, adam),\\ &male(adam), male(abel), female(eva),\\ &(\forall x, y)((child(y, x) \wedge male(x)) \rightarrow father(x, y))\} \end{aligned}$$

Cíli `(?` ψ_1 ... ψ_n`)` odpovídá formule $\varphi = \psi_1 \wedge \cdots \wedge \psi_n$. Například cíli `(? (father adam abel))` odpovídá formule $\varphi = father(adam, abel)$.

Struktura M je **modelem** T, jestliže je každá formule z T v M pravdivá. Uvažujme dvě struktury:

$$\begin{aligned} M_1 =&\{0, 1, 2\}, adam^{M_1} = 0, eva^{M_1} = 1, abel^{M_1} = 2,\\ &child^{M_1} = \{\langle 2, 1\rangle, \langle 2, 0\rangle\}, male^{M_1} = \{0, 2\}, female^{M_1} = \{1\}, father^{M_1} = \{\langle 0, 2\rangle\};\\ M_2 =&\{0, 1, 2\}, adam^{M_2} = 0, eva^{M_2} = 1, abel^{M_2} = 2,\\ &child^{M_2} = \{\langle 2, 1\rangle, \langle 2, 0\rangle\}, male^{M_2} = \{0, 2\}, female^{M_2} = \{1\}, father^{M_2} = \{\langle 0, 1\rangle\}. \end{aligned}$$

Struktura M_1 je modelem teorie T, ale struktura M_2 modelem teorie T není.

Zajímavé je, že každé teorie programu má model. Například pro výše uvedenou teorii můžeme uvažovat model:

$$\begin{aligned} M_3 =&\{0\}, adam^{M_3} = eva^{M_3} = abel^{M_3} = 0,\\ &child^{M_3} = father^{M_3} = \{\langle 0, 0\rangle\}, male^{M_3} = female^{M_3} = \{0\} \end{aligned}$$

Také můžeme vytvořit model, kde univerzum (nosná množina) budou termy. Například:

$$\begin{aligned} M_4 =& \{adam, eva, abel\}, adam^{M_4} = adam, eva^{M_4} = eval, abel^{M_4} = abel, \\ & child^{M_4} = \{\langle abel, eva\rangle, \langle abel, adam\rangle\}, male^{M_4} = \{adam, abel\}, \\ & female^{M_4} = \{eva\}, father^{M_4} = \{\langle adam, abel\rangle\} \end{aligned}$$

Formule φ **sémanticky vyplývá** z T, jestliže je φ pravdivá v každém modelu T. Zapisujeme $T \vDash \varphi$. Platí:

Cíl φ je vzhledem k T splněný, právě když $T \vDash \varphi$.

Například:

$$\begin{aligned} &\{child(abel, eva), child(abel, adam), \\ &\ male(adam), male(abel), female(eva), \\ &\ (\forall x, y)((child(y, x) \wedge male(x)) \rightarrow father(x, y))\} \vDash father(adam, abel) \end{aligned}$$

5 Poznámky k implementaci

Nově se logický program udržuje v globální proměnné `*rules*`. K úpravě pravidel programu slouží makro `defpredicate` se syntaxí

```
(defpredicate rules)
```

kde ***rules*** jsou pravidla, jejichž hlavy mají shodný predikát. Například:

```
(defpredicate
 (<- (child abel eva))
 (<- (child abel adam)))
```

Makro nejprve odebere všechna pravidla pro určený predikát a poté přidá do seznamu `*rules*` pravidla ***rules***.

Ukázkový program můžeme definovat následovně.

```
(defpredicate
 (<- (child abel eva))
 (<- (child abel adam)))
```

```
(defpredicate
 (<- (male adam))
 (<- (male abel)))

(defpredicate
 (<- (female eva)))

(defpredicate
 (<- (father ?x ?y) (child ?y ?x) (male ?x)))
```

Makro `?` se syntaxí

```
(? subgoal1 subgoal2 ...)
```

rozhodne, zda je cíl `(? subgoal1 subgoal2 ...)` vzhledem k `*rules*` splněný.

Funkce `view-resolution-tree` se syntaxí

```
(view-resolution-tree goal)
```

zobrazí strom úsudků pro cíl *goal* vzhledem k programu `*rules*`.

Otázky a úkoly na cvičení

1. Rozhodněte, zda první výraz je instancí druhého. V kladném případě uveďte jejich unifikátor.
   (a) `(succ (succ zero))` a `(succ ?n)`
   (b) `(+ zero (succ zero) (succ (succ zero)))` a `(+ zero ?x ?x)`
   (c) `(double (succ zero) (succ (succ (succ zero))))` a `(double (succ ?x) (succ (succ ?y)))`
2. Uvažujme predikát:

```
(defpredicate
 (<- (member ?x (?x . ?xs)))
 (<- (member ?x (?y . ?ys))
     (member ?x ?ys)))
```

   Zakreslete strom úsudků pro cíle:

   - `(? (member b (a b c)))`

- `(? (member d (a b c)))`

3. Celá nezáporná čísla reprezentujeme následujícími termy. Konstanta `zero` je nula, term `(succ` ***n***`)`, kde ***n*** je term zapisující číslo, je následník čísla zapsaného termem ***n***. Například číslo dva zapíšeme termem `(succ (succ zero))`. Definujte predikát `number`, který rozhodne, zda je argument objekt zapisující číslo. Tedy cíl `(? (number (succ zero)))` by měl být splněn, ale cíl `(? (number (succ a)))` ne.

4. Pro čísla zapsaná pomocí termů definujte následující predikáty.

   (a) `(even` ***n***`)` rozhodne, zda ***n*** je sudé číslo.

   (b) `(double` ***m n***`)` rozhodne, zda ***n*** je dvojnásobek ***m***.

   (c) `(+` ***m n k***`)` rozhodne, zda ***k*** je součet ***m*** a ***n***.

   (d) `(<=` ***m n***`)` rozhodne, zda ***m*** je menší nebo rovno ***n***.

   (e) `(<` ***m n***`)` rozhodne, zda ***m*** je ostře menší než ***n***.

5. Napište predikát `list`, který rozhodne, zda je jediný argument čistý seznam. Například cíl `(? (list (a b)))` by měl být splněn, ale cíle `(? (list (a . b)))` a `(? (list a))` by splněny být neměly.

6. Definujte pro seznamy následující predikáty.

   (a) `(length` ***list n***`)` rozhodne, zda ***list*** je seznam délky ***n***.

   (b) `(subset` ***set1 set2***`)` rozhodne, zda seznam ***set2*** obsahuje všechny prvky seznamu ***set1***. Například cíl `(? (subset (a b b) (b a c)))` má být splněn.

   (c) `(append` ***list1 list2 list3***`)` rozhodne, zda seznam ***list3*** je rovný spojení seznamů ***list1*** a ***list2***.

   (d) `(rev-append` ***list1 list2 list3***`)` rozhodne, zda seznam ***list3*** je rovný spojení seznamů ***list1*** a ***list2***, kde prvky seznamu ***list1*** jsou převráceny. Například `(? (rev-append (a b) (c d) (b a c d)))` má být splněn.

   (e) `(reverse` ***list1 list2***`)` rozhodne, zda list ***list2*** je rovný převrácení seznamu ***list1***.

   (f) `(palindrome` ***list***`)` rozhodne, zda ***list*** se čte stejně zepředu i pozpátku. Například cíl `(? (palindrome (a b b a)))` má být splněn.