
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**

# 9. Usuzování s výroky

verze z 8. dubna 2025

## 1 Motivační příklad

Začněme motivačním příkladem. Představme si, že vytváříme bezpečnostní systém, jehož chování je dáno pravidly:

1. Jestliže je zamčeno a je detekován pohyb, pak je alarm spuštěný.
2. Jestliže je detekován kouř, pak je alarm spuštěný.

Všimněme si, že pravidla mají tvar implikace, kde předpoklad implikace je konjunkce atomických výroků a závěr implikace je jediný atomický výrok. Náš bezpečností systém má čidla, která mohou detekovat fakta. Například:

1. Je zamčeno.
2. Je detekován kouř.

Fakta jsou atomické výroky. Rozhodnutí, zda spustit alarm, je dáno dotazem: Je alarm spuštěný? Dotaz může obecně být konjunkce atomických výroků.

## 2 Logický program

Nejprve vyřešíme, jak v programu reprezentovat atomické výroky. **Atomický výrok** reprezentujeme necyklickou strukturou tečkových párů. Zatím si vystačíme pouze se symboly:

1. alarm je spuštěný ... `alarm-on`
2. místnost je zamčená ... `room-locked`
3. je detekován pohyb ... `move-detected`
4. je detekován kouř ... `smoke-detected`

**Pravidlo** reprezentujeme seznamem `(<- head . body)`, kde

- *`head`* je atomický výrok (**hlava pravidla**),

- *body* je seznam atomických výroků (**tělo pravidla**).

Reprezentace pravidel našeho systému:

1. Jestliže je zamčeno a je detekován pohyb, pak je alarm spuštěný.
   ... `(<- alarm-on room-locked move-detected)`
2. Jestliže je detekován kouř, pak je alarm spuštěný.
   ... `(<- alarm-on smoke-detected)`

Konjunkce prázdné množiny výroků je vždy splněna. **Fakt** tedy můžeme reprezentovat pravidlem s prázdným tělem. Reprezentace faktů z motivačního příkladu:

1. Je zamčeno. ... `(<- room-locked)`
2. Je detekován kouř. ... `(<- smoke-detected)`

Dotaz formujeme jako cíl. **Cíl** je seznam `(?` *subgoal1 subgoal2 ...*`)`, kde *subgoal1*, *subgoal2* ... jsou atomické výroky (**podcíle**). Například výrok „Je alarm spuštěný?“ reprezentujeme seznamem `(? alarm-on)`. Cíl bez výroků nazýváme **prázdný cíl**.

**Logický program** je seznam pravidel. Bezpečnostní systém v určitém čase:

```
((<- alarm-on room-locked move-detected)
 (<- alarm-on smoke-detected)
 (<- room-locked)
 (<- smoke-detected))
```

# 3 Usuzování

Podívejme se nejprve na jeden krok usuzování, který budeme nazývat úsudek.

Na cíl:

- `(?` *subgoal1 subgoal2 ...*`)`

lze použít pravidlo:

- `(<-` *head body1 ... bodyn*`)`

pokud *subgoal1* je rovno *head*. Použitím pravidla na cíl obdržíme cíl, kde v původním cíli nahradíme první podcíl za tělo pravidla:

- `(?` *body1 ... bodyn subgoal2 ...*`)`

Například:

1. Použitím pravidla `(<- alarm-on room-locked move-detected)` na `(? alarm-on)` obdržíme `(? room-locked move-detected)`.
2. Použitím pravidla `(<- room-locked)` na `(? room-locked move-detected)` obdržíme `(? move-detected)`.

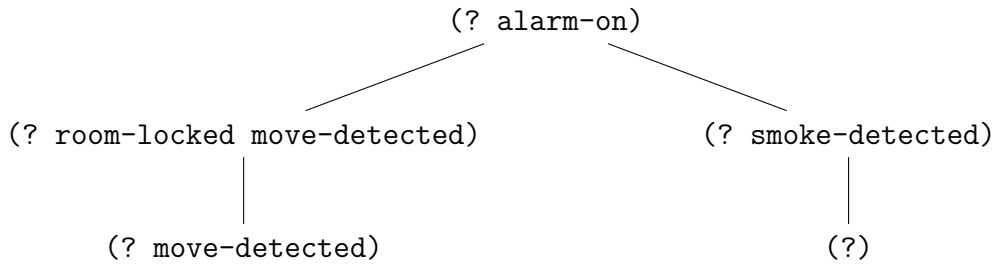
**Strom úsudků** (také **rezoluční strom**) pro dotaz ***query*** vzhledem k programu ***rules*** je strom, kde platí následující.

- Uzly stromu jsou ohodnoceny cíli.
- Kořen stromu má cíl ***query***.
- Uzel stromu má tolik následníků, kolik pravidel z ***rules*** lze použít na jeho cíl.
- Pokud lze použít pravidlo ***rule*** z ***rules*** na cíl ***goal*** uzlu stromu, pak má uzel následníka s cílem rovným výsledku použití pravidla ***rule*** na ***goal***.

Například uvažujme program:

```
((<- alarm-on room-locked move-detected)
 (<- alarm-on smoke-detected)
 (<- room-locked)
 (<- smoke-detected))
```

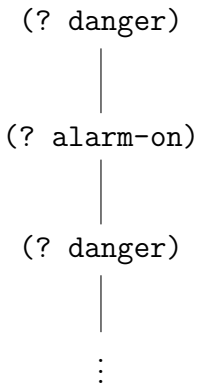
Pak strom pro dotaz `(? alarm-on)` je:




Vezměme jiný program:

```
((<- danger alarm-on)
 (<- alarm-on danger))
```

Strom úsudků pro `(? danger)` je nekonečný:




Cíl *query* je vzhledem k programu *rules* **splněn**, jestliže jeho strom úsudků obsahuje prázdný cíl. Například cíl `(? alarm-on)` je vzhledem k

```
((<- alarm-on room-locked move-detected)
 (<- alarm-on smoke-detected)
 (<- room-locked)
 (<- smoke-detected))
```

splněný, ale `(? room-locked move-detected)` není. Dále cíl `(? danger)` vzhledem k

```
((<- danger alarm-on)
 (<- alarm-on danger))
```

splněný není.

# 4 Implementace

Cíl reprezentujeme tečkovým párem `(? . subgoals)`, kde *subgoals* je seznam podcílů. Konstruktor cíle a jeho selektory:

```
(defun goal (subgoals)
  (cons '? subgoals))

(defun goal-subgoals (goal)
  (cdr goal))

(defun empty-goal-p (goal)
  (null (goal-subgoals goal)))
```

Pravidlo reprezentujeme seznamem `(<- head . body)`, kde *head* je hlava pravidla a *body* je seznam tvořící tělo pravidla. Konstruktor pravidla a jeho selektory:

```
(defun rule (head body)
  `(<- ,head ,@body))

(defun rule-head (rule)
  (cadr rule))

(defun rule-body (rule)
  (cddr rule))
```

Následující predikát rozhoduje, zda lze pravidlo použít na cíl.

```
(defun applicable-rule-p (rule goal)
  (and (not (empty-goal-p goal))
       (equalp (rule-head rule)
               (car (goal-subgoals goal)))))
```

Výsledek použití pravidla vrací funkce:

```
(defun rule-resolvent (rule goal)
  (goal (append (rule-body rule)
                (cdr (goal-subgoals goal)))))
```

Následující funkce vrací seznam všech pravidel, která jsou použitelná na zadaný cíl.

```
(defun applicable-rules (goal rules)
  (remove-if-not (lambda (rule)
                   (applicable-rule-p rule goal))
                 rules))
```

Dále budeme potřebovat funkci, která vrátí seznam cílů obdržených ze zadaného cíle použitím všech použitelných pravidel.

```
(defun goal-resolvents (goal rules)
  (mapcar (lambda (rule)
            (rule-resolvent rule goal))
          (applicable-rules goal rules)))
```

Uzel stromu reprezentujeme seznamem `(node` *`value . children`*`)`, kde *`value`* je hodnota uzle a *`children`* seznam jeho následníků. Konstruktor uzlu stromu a jeho selektory:

```
(defun tree-node (value children)
  (cons 'node (cons value children)))

(defun node-value (node)
  (cadr node))

(defun node-children (node)
  (cddr node))
```

Strom úsudků pro zadaný cíl vzhledem k programu vytvoříme rekurzivně:

```
(defun resolution-tree (goal rules)
  (tree-node goal
             (mapcar (lambda (g)
                       (resolution-tree g rules))
                     (goal-resolvents goal rules))))
```

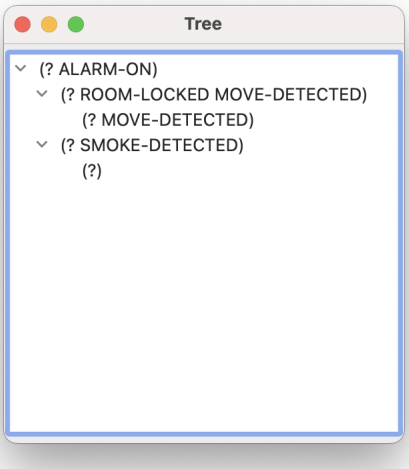
Například:

```
CL-USER 1 > (resolution-tree '(? alarm-on)
                  '((<- alarm-on room-locked move-detected)
                    (<- alarm-on smoke-detected)
                    (<- room-locked)
                    (<- smoke-detected)))
(NODE (? ALARM-ON)
      (NODE (? ROOM-LOCKED MOVE-DETECTED)
            (NODE (? MOVE-DETECTED)))
      (NODE (? SMOKE-DETECTED)
            (NODE (?))))
```

Funkci `(view-tree` *`tree`*`)` můžeme použít k zobrazení libovolného stromu, tedy i rezolučního. Například:

```
CL-USER 2 > (view-tree *)
```

otevře okno zobrazující předchozí strom:




Pro nalezení prázdného cíle můžeme použít průchod do hloubky:

```
(defun tree-values-dfs (root)
  (cons (node-value root)
        (reduce #'append
                (mapcar #'tree-values-dfs
                        (node-children root))
                :initial-value '())))
```

nebo do šířky:

```
(defun tree-values-bfs (root)
  (tvb (list root)))

(defun tvb (roots)
  (if (null roots)
      '()
    (append (mapcar #'node-value roots)
            (tvb (reduce #'append
                         (mapcar #'node-children roots)
                         :initial-value '())))))
```

Průchod vrací seznam hodnot uzlů stromu.

Rozhodnutí o splnění cíle provedeme tak, že nejprve vytvoříme rezoluční strom, projdeme jej do hloubky, nebo do šířky. Nyní je cíl splněn, právě když průchod obsahuje prázdný cíl.

```
(defun fulfilledp (goal rules &optional breadthp)
  (find-if #'empty-goal-p
           (funcall (if breadthp
                        #'tree-values-bfs
                      #'tree-values-dfs)
                    (resolution-tree goal rules))))
```

Samozřejmě funkce neskončí, jestliže pravidla a cíl vedou k nekonečnému rezolučnímu stromu. Například:

```
(fulfilledp '(? danger)
            '((<- danger alarm-on)
              (<- alarm-on danger)))
```

# 5 Logický pohled

Ve výrokové logice reprezentujeme výroky výrokovými symboly. Příklad:

1. alarm je spuštěný ... $p_a$
2. místnost je zamčená ... $p_l$
3. je detekován pohyb ... $p_m$
4. je detekován kouř ... $p_s$

Pravidlo `(<-` $p\ q_1\ \ldots\ q_n$`)` je formule $(q_1 \wedge \cdots \wedge q_n) \rightarrow p$. Příklad:

1. Jestliže je zamčeno a je detekován pohyb, pak je alarm spuštěný.
   ... $(p_l \wedge p_m) \rightarrow p_a$
2. Jestliže je detekován kouř, pak je alarm spuštěný.
   ... $p_s \rightarrow p_a$

Fakt `(<-` $p$`)` je formule $(q_1 \wedge \cdots \wedge q_0) \rightarrow p = 1 \rightarrow p \equiv p$. Tedy přímo výrokový symbol $p$. Symbolem $\equiv$ značíme sémantickou rovnost formulí. Příklad:

1. Místnost je zamčená ... $p_l$
2. Je detekován kouř ... $p_s$

Program je konečná množina formulí $T$ (**teorie**). Příklad:

$$T = \{(p_l \wedge p_m) \rightarrow p_a, p_s \rightarrow p_a, p_l, p_s\}$$

Cíl `(? `$p_1$ `...` $p_n$`)` je formule $\varphi = p_1 \wedge \cdots \wedge p_n$. Například cíl „Je alarm spuštěný?“ reprezentujeme ve výrokové logice formulí $\varphi = p_a$.

Fakt, že formule $\varphi$ sémanticky vyplývá z $T$ značíme $T \vDash \varphi$. Chceme výpočtem rozhodnout, zda $T \vDash \varphi$. Příklad:

$$\{(p_l \wedge p_m) \rightarrow p_a, (p_l \wedge p_s) \rightarrow p_a, p_l, p_s\} \models p_a$$

Použitím pravidla $(q_1 \wedge \cdots \wedge q_n) \rightarrow p_1 \in T$ na cíl $\varphi_1 = p_1 \wedge \cdots \wedge p_m$ obdržíme cíl $\varphi_2 = q_1 \wedge \cdots \wedge q_n \wedge p_2 \wedge \cdots \wedge p_m$.

Pokud $T \vDash \varphi_2$, pak $T \vDash \varphi_1$. Skutečně. Předpokládejme, že $T \vDash \varphi_2$ a vezměme ohodnocení výrokových proměnných $e$ takové, že $||T||_e = 1$ (každá formule z $T$ je v $e$ pravdivá). Protože $T \vDash \varphi_2$, pak $||\varphi_2||_e = 1$ (formule $\varphi_2$ je v $e$ pravdivá). Tedy $||q_1||_e = \cdots = ||q_n||_e = ||p_2||_e = \cdots = ||p_m||_e = 1$. Protože $(q_1 \wedge \cdots \wedge q_n) \rightarrow p_1 \in T$, platí $||(q_1 \wedge \cdots \wedge q_n) \rightarrow p_1||_e = 1$. Tedy i $||p_1||_e = 1$. Dostáváme, že $||p_1 \wedge \cdots \wedge p_m|| = 1$. Ukázali jsme, že $T \vDash \varphi_1$.

Z předchozího tvrzení plyne, že pokud je cíl $\varphi$ je vzhledem k $T$ splněný, pak $T \vDash \varphi$. Platí i obrácená implikace, kterou zde dokazovat nebudeme. Tedy dohromady platí:

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

Tedy predikát `fulfilledp` rozhoduje, zda dotaz sémanticky vyplývá z programu.

# Otázky a úkoly na cvičení

1. Úkoly se nalézají v připojeném souboru `09_tic_tac_toe_tutorial.lisp`.