Cvičení 1

Programování s omezujícími podmínkami

Roman Barták
Katedra teoretické informatiky a matematické logiky

`roman.bartak@mff.cuni.cz`
`http://ktiml.mff.cuni.cz/~bartak`

Využití CSP

- Využití principů, ale **vlastní naprogramování** řešících algoritmů.
  - flexibilita (možnost plně přizpůsobit vlastním potřebám)
  - rychlost (pro daný problém)
  - náročné na počáteční vývoji i údržbu
- Použití **existujícího řešiče** podmínek.
  - zpravidla integrovaný formou knihovny do hostitelského jazyka
  - naprogramované řešící algoritmy
  - soustředění na modelování problému
  - nelze zasahovat do nízko-úrovňových struktur (definice domén,…)
  - někdy možnost definovat vlastní podmínky
  - často možnost definovat vlastní prohledávání

Řešiče podmínek

`http://ktiml.mff.cuni.cz/~bartak/constraints/systems.html`

- Poskytované služby:
  - implementace **datových struktur** pro modelování domén proměnných a pro podmínky
  - základní rámec **propagace podmínek**
  - **filtrační algoritmy** pro řadu podmínek (včetně globálních podmínek)
  - základní **prohledávací strategie** včetně heuristik pro výběr proměnných a hodnot
  - **rozhraní** pro definici vlastních podmínek
- Klasifikace:
  - samostatné řešiče
    - Minion
  - vlastní programovací/modelovací jazyk
    - Mozart, OPL, Comet, CHR
  - hostitelský jazyk
    - **Prolog**: ECLiPSe, SICStus Prolog
    - **C/C++**: ILOG Solver, Gecode
    - **Java**: Choco, JaCoP

SICStus Prolog

`http://www.sics.se/sicstus`

- Komerční systém, 30-denní zkušební licence
- **Vlastnosti**
  - podpora ISO standardu Prologu
  - podpora řady počítačových platforem (Win, MacOS X, Linux, Solaris)
  - vývojové prostředí GNU Emacs
  - řada knihoven včetně clpfd
  - možnost tvorby samostatných (stand-alone) i vestavěných (embedded) aplikací
- **Proč Prolog?**
  - jednoduchá syntax
  - krátký program hodně umí
  - přímočará integrace omezujících podmínek
  - prohledávání je součástí řešícího systému

SICStus 4

Základní koncept Prologu

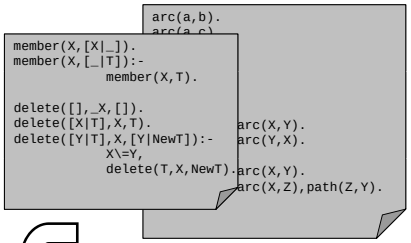
Prolog je deduktivní systém, který hledá odpovědi na **otázky** použitím báze znalostí skládající se z **faktů** a **odvozovacích pravidel**.

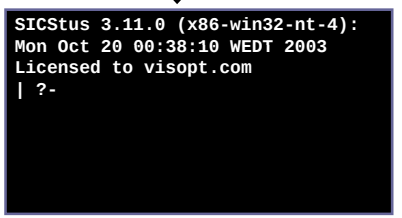
Kde je programování?

- tvorba databáze faktů a pravidel
- Prolog automaticky odvozuje odpovědi
- ↳ **deklarativní programování**

Architektura Prologu

- Zdrojové soubory
  - *.pl
- Prologvská databáze
- Dotazy





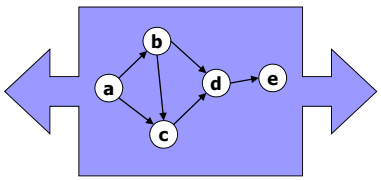
Prologovské fakty

Prologovské fakty popisují základní údaje o řešeném problému.

```
node(a).
node(b).
node(c).
node(d).
node(e).
```

jméno

argument




```
arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```

více argumentů se odděluje čárkami

Přímé dotazy

Je možné klást **dotazy** na fakty uložené v bázi znalostí Prologu:

Prolog prompt

dotaz

odpověď

```
?-node(a).
yes
?-node(bla).
no
?-arc(a,c).
yes
?-arc(a,d).
no
?-path(a,d).
no
```

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```

Otevřené dotazy

Dotaz může obsahovat **proměnné**, jejichž hodnoty budou nalezeny v bázi znalostí:

```
?-node(X).
X=a ;
X=b ;
X=c ;
X=d ;
X=e ;
no

?-arc(a,X).
X=b ;
X=c ;
no
```

požadavek na alternativní odpověď

žádné další odpovědi

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```


Složené dotazy

- Seznam faktů je vlastně jednoduchá databáze.
- Je možné zodpovědět otázku, jejíž odpověď není přímo uložena v databázi faktů, ale kterou lze získat kombinací faktů?

Ano, je možné klást **dotaz o kombinaci faktů** z báze znalostí:

```
?-arc(a,Y),arc(Y,Z).
Y=b
Z=c ;
Y=b
Z=d ;
Y=c
Z=d ;
no
```

Proměnné je možné sdílet mezi přímými dotazy

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```


Syntatická pauza

atomy vs. proměnné

Data (a programy) jsou vyjádřeny formou **termů**.

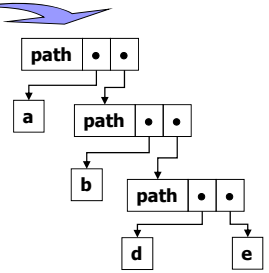
- **Atomy**
  - slova skládající se z písmen, čísel a podtržítek, která začínají malým písmenem
    - `a, arc, john_123, …`
  - slova v jednoduchých uvozovkách
    - `´Edinburgh´, …`
- **Proměnné**
  - slova skládající se z písmen, čísel a podtržítek, která začínají velkým písmenem nebo podtržítkem
    - `X, Node, _noname, …`
  - _ je tzv. anonymní proměnná
    - různé výskyty _ jsou brány jako různé proměnné
    - obsah proměnné není uživateli přístupný

Syntaktická pauza

složené termy

Složené termy vyjadřují **strukturovanou informaci**

- atomy a proměnné jsou jednoduché termy
- **functor(arg1,…,argn)** je složený term, kde functor je atom a arg1, …, argn jsou termy
  - arc(a,c)
  - **path(a,path(b,path(d,e)))**
  - tree(tree(a,tree(b,c)),tree(d,e))
  - arc(a,X)
  - …



Odvozovací pravidla

- dotaz můžeme pojmenovat a uložit pro opakované použití

```
doubleArc(X,Z):-arc(X,Y),arc(Y,Z).
```

  - Takovému dotazu se říká **odvozovací pravidlo**.
- Po formulaci pravidla na něj můžeme klást dotazy stejně jako na fakta:

```
?-doubleArc(b,W).
W=d ;
W=e ;
no
?-doubleArc(a,W).
W=c ;
W=d ;
W=d ;
no
```

pouze proměnné z názvu pravidla jsou vráceny uživateli

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```


Jak to funguje?

odvozovací pravidla

```
?-doubleArc(b,W).
```

- najdi pravidlo, jehož název odpovídá dotazu a příslušným způsobem navaž proměnné

```
doubleArc(b,W):-arc(b,Y),arc(Y,W).
```

- nahraď dotaz definicí pod-dotazu z pravidla

```
?-arc(b,Y),arc(Y,W).
```

- najdi odpovídající fakt (`arc(b,c)`), dosaď proměnné a odstraň z dotazu

```
?-arc(c,W).
```

- opakuj to samé se zbytkem (`arc(c,d)`)

```
W=d ;
```

- zkus alternativní fakta (`arc(b,d),arc(d,e)`)

```
W=e ;
no
```

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```


Alternativní pravidla

- Je možné také definovat **alternativní pravidla** (disjunkce dotazů)

```
edge(X,Y):-arc(X,Y).
edge(X,Y):-arc(Y,X).
```

```
?-edge(W,b).
W=a ;
W=c ;
W=d ;
no
```

odvozeno pomocí prvního pravidla

odvozeno pomocí druhého pravidla

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```


Jak to funguje?

alternativní pravidla

Stejně jako předtím, pouze se zkouší i alternativní pravidla.

```
?-edge(W,b).
```

- Najdi pravidlo, jehož název odpovídá dotazu, navaž proměnné a nahraď dotazem definicí dotazu z pravidla

```
edge(W,b):-arc(W,b).
```

```
?-arc(W,b).
```

- najdi řešení dotazu pomocí faktů

```
W=a ;
```

- zkus alternativní pravidlo pro původní dotaz

```
edge(W,b):-arc(b,W).
```

```
?-arc(b,W).
```

- najdi řešení dotazu pomocí faktů

```
W=c ;
W=d ;
no
```

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```

Rekurzivní pravidla

- Je dokonce možné použít jméno pravidlo při definici pod-dotazu, tj. použít **rekurzi**

```
path(X,Y):-arc(X,Y).
path(X,Y):-arc(X,Z),path(Z,Y).
```

```
?-path(c,W).
W=d ;
W=e ;
no
```

odvozeno prvním pravidle a arc(c,d)

odvozeno druhým pravidlem a rekurzí

```
node(a).
node(b).
node(c).
node(d).
node(e).

arc(a,b).
arc(a,c).
arc(b,c).
arc(b,d).
arc(c,d).
arc(d,e).
```

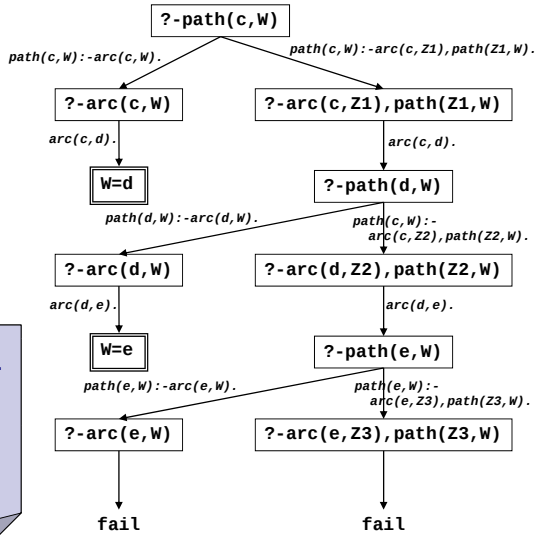

Jak to funguje?

rekurzivní pravidla

- Stejně jako předtím, ale pravidlo může být použito vícekrát.
- Každé užití pravidla vyžaduje **čerstvou kopii proměnných** (podobné jako volání procedury s lokálními proměnnými).

```
path(X,Y):-arc(X,Y).
path(X,Y):-arc(X,Z),path(Z,Y).

node(a).      arc(a,b).
node(b).      arc(a,c).
node(c).      arc(b,c).
node(d).      arc(b,d).
node(e).      arc(c,d).
              arc(d,e).
```



Prolog ve zkratce

Prologovský **„program"** se skládá z **pravidel** a **faktů.**
Každé **pravidlo** má strukturu **Hlava:-Tělo.**

- **Hlava** je (složený) term
- **Tělo** je dotaz (konjunkce termů)
  - Tělo typicky obsahuje všechny proměnné z Hlavy
- sémantika: **pokud je Tělo pravda potom odvoď Hlavu**

Fakty jsou vlastně pravidla s prázdným tělem (true).

Dotaz je konjunkce termů: Q = **Q1,Q2,…,Qn.**

- **Najdi pravidlo**, jehož hlava odpovídá dotazu Q1.
  - Pokud je takových pravidel více, vytvoř „bod volby" a použij první pravidlo.
  - Pokud neexistuje žádné pravidlo, potom se vrať na poslední bod volby a zkus zde alternativní pravidlo.
- **Použij tělo pravidla**, kterým v dotazu nahraď část Q1.
  - Pro pravidla, kde Tělo=true, dotaz Q1 zmizí.
- **Opakuj, dokud nezískáš prázdný dotaz**.

Technologie Prologu

Prolog = Unifikace + Backtracking

- **Unifikace** (matching) slouží pro
  - výběr vhodného pravidla (dle hlavy)
  - vytvoření odpovědní substituce
  - Jak?
    - substitucí se snaží udělat termy syntakticky identické
- **Backtracking** (prohledávání do hloubky) je pro
  - prozkoumání alternativ
  - Jak?
    - nejdříve vyřeš první část (zleva) dotazu
    - použij první pravidlo (shora)