Cvičení 4

Programování s omezujícími podmínkami

Roman Barták
Katedra teoretické informatiky a matematické logiky

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

Pro připomenutí

- Typická struktura programu CLP:

```
:-use_module(library(clpfd)).

solve(Sol):-
  declare_variables( Variables),
  post_constraints( Variables ),
  labeling( Variables ).
```

definice operátorů CLP, podmínek a řešících strategií

definice proměnných a jejich domén

definice omezení

deklarativní model problému

Řídící část
- prohledávání prostoru možných řešení
- výběr hodnot proměnných
- najde jedno, všechna nebo optimální řešení

Definice domén

- doména v SICStus Prologu je množina celých čísel
  - jiné hodnoty je potřeba mapovat na čísla
  - čísla mají přirozené uspořádání
- doména je často interval
  - `domain(SeznamProměnných,MinVal,MaxVal)`
  - definuje proměnné s počáteční doménou {MinVal, …,MaxVal}
- doménu můžeme definovat pro proměnné samostatně (možno i sjednocení, průnik, doplněk intervalů)
  - `X in MinVal..MaxVal`
  - `X in (1..3) \/ (5..8) \/ {10}`

Reprezentace domén

- domény jsou reprezentovány jako seznam disjunktních intervalů
  - $[[Min_1|Max_1],[Min_2|Max_2],\ldots,[Min_n|Max_n]]$
  - $Min_i \leq Max_i < Min_{i+1} - 1$
- definice domény odpovídá unární podmínce
  - tj. je-li pro stejnou proměnnou definováno více domén, bere se jejich průnik (jako konjunkce unárních podmínek)

```
?-domain([X],1,20), X in 15..30.
X in 15..20
```

Specifikace podmínek
aritmetické

- klasické aritmetické podmínky s operacemi +,-, *, /, abs, min, max,… operace jsou vestavěné
- samozřejmě je potřeba použít nějaký porovnávací operátor #=, #<, #>, #=<, #>=, #\=

```
?-A+B #=< C-2.
```

- Co když definuji podmínku dříve než doménu?
  - pro nové proměnné z takové podmínky se bere nekonečná doména inf..sup

Jak to funguje?

Jak je realizováno splňování podmínek?

- Pro každou proměnnou systém udržuje aktuální doménu.
- Jakmile je přidána nějaká podmínka, nekonzistentní hodnoty jsou odstraněny z domén proměnných.

Příklad:

	X	Y
	inf..sup	inf..sup
domain([X,Y],0,100)	0..100	0..100
3#=X+Y	0..3	0..3
Y#>=2	0..1	2..3
X#>=1	1	2

Vlastní složené podmínky
all_diff

- Napište program „definující" podmínku **all_diff(List).**
  - proměnné v seznamu List jsou navzájem různé.

```
all_diff([]).
all_diff([H|T]):
   diff_from(T,H),!,all_diff(T).

diff_from([],_).
diff_from([H|T],X):-
   X #\= H,!,diff_from(T,X).
```


Specifikace podmínek
logické

Aritmetické (reifikovatelné) podmínky můžeme spojovat do složitějších logických konstrukcí pomocí logických spojek:

- `#\ :Q` negace
- `:P #/\ :Q` konjunkce
- `:P #\ :Q` exklusivní disjunkce („právě jedna")
- `:P #\/ :Q` disjunkce
- `:P #=> :Q` implikace
- `:Q #<= :P` implikace
- `:P #<=> :Q` ekvivalence

```
?- X#<5 #\/ X#>8.
X in inf..sup
```

Disjunkce

Začněme jednoduchým příkladem

```
:-use_module(library(clpfd)).

a(X):- X#<5.
a(X):- X#>7.
```

```
?- a(X).
 {SYNTAX ERROR: in
X in inf..sup ?
** operator ? expected
 after expression **
no
```

Kde je problém?

Model je disjunktivní, tj. potřebujeme „backtrackovat“, abychom získali model, kde je X>7!

```
:-use_module(library(clpfd)).

a(X):- X#<5 #\/ X#>7.
```

```
?- a(X).

X in inf..sup ? ;

no
```

```
?- a(X), X#>5.

X in 8..sup ? ;

no
```

Propagátor čeká dokud všechny členy disjunkce až na jeden jsou nepravdivé a potom propaguje přes poslední člen.

Konstruktivní disjunkce

```
:-use_module(library(clpfd)).

a(X):- X in (inf..4) \/ (8..sup).
```

```
?- a(X).

X in (inf..4)\/(8..sup) ? ;

no
```

Konstruktivní disjunkce

Jak to funguje v obecném případě?

$a_1(X) \vee a_2(X) \vee \ldots a_n(X)$

propaguj každou podmínku $a_i(X)$ odděleně

sjednoť domény pro X z těchto propagací

To může být pěkně drahé!

Podobné bodové konzistenci:

$X \text{ in } 1..5 \Rightarrow X=1 \vee X=2 \vee X=3 \vee X=4 \vee X=5$

Pro konkrétní disjunktivní podmínky ale můžeme psát speciální propagátory!

Specifikace podmínek
tabulkové

- Co když potřebujeme popsat obecnou podmínku – n-ární relaci?
- Konkrétně, jak popsat podmínku zadanou množinou kompatibilních hodnot?
- **`table(Vars,Extension)`**

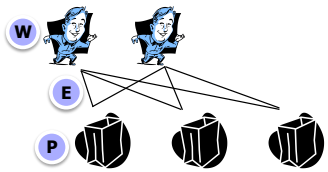
```
?-table([[X,Y,Z]],[[1,2,3],[1,3,4]]).
X = 1,
Y in 2..3,
Z in 3..4
```

Je možné popsat stejnou podmínku zároveň nad více n-ticemi hodnot.

```
table([[X,Y,Z],[P,Q,R]], ...)
```


Modelování

Uvažujme dva pracovníky, kteří mohou pracovat na třech produktech, kde efektivita pracovníka při výrobě produktu je dána tabulkou.



	P1	P2	P3
W1	7	1	3
W2	8	2	5

Navrhněte podmínku(y), která svazuje proměnné W (pracovník), P (produkt) a E (efektivita).

```
W in {1,2}, P in 1..3,
table([[W,P,E]], [[1,1,7),[1,2,1],[1,3,3],
[2,1,8],[2,2,2],[2,3,5]])
```

Instanciace proměnných

- Podmínky většinou nezajistí ohodnocení proměnných a proto potřebujeme přiřadit hodnoty proměnným – prohledávání.
- **`indomain(X)`**
  - do proměnné přiřadí hodnotu (hodnoty zkouší v rostoucím pořadí)
- **`labeling(Params,Vars)`**
  - ohodnotí proměnné Vars
  - Algoritmus MAC – backtracking s udržováním hranové konzistence

Příklad

- Nalezněte možná řešení rovnice
  $A + B = 10$ pro $A, B \in \{1, 2, \ldots, 10\}$

```
:- use_module(library(clpfd)).
aritmetika(A,B) :-
  domain([A,B], 1, 10),
  A + B #= 10,
  labeling([],[A,B]).
```


Příklad

- Nalezněte možná řešení Pythagorovy věty
  $A^2 + B^2 = C^2$ $(A, B, C \in \{1, \ldots, 20\})$

```
:- use_module(library(clpfd)).
pythagoras(A,B,C) :-
  domain([A,B,C], 1, 20),
  A*A + B*B #= C*C,
  A #=< B, % odstranění
  symetrických řešení
  labeling([],[A,B,C]).
```


Domácí úkol

- Vyřešte algebrogram
  DONALD + GERARD = ROBERT použitím modelu s přenosovými bity.