Cvičení 9

Programování s omezujícími podmínkami

Roman Barták
Katedra teoretické informatiky a matematické logiky

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

Zápočtové programy

- Zdrojový kód spustitelný pod SICStus Prolog 4
- "Dokumentace" (1-2 stránky)
  - zadání problému
  - popis vstupů, spuštění programu a výstupu
    - program by typicky měl umět řešit problémy daného typu (ne pouze jeden konkrétní problém)
  - abstraktní popis modelu s podmínkami
  - experimentální výsledky
    - srovnání různých modelů, problémů různých velikostí
    - runtime, počet bodů volby apod.

Zápočtové programy

- hledání cest v grafu pro více agentů
- hledání cesty pro "uklízecího" robota
- Zebra
- generování Sudoku
- obecné rozvrhování s unárními zdroji (JSSP)
- dinner party problem
- peaceable armies of queens
- open stacks problem
- 2D protein folding
- …

Další program

Podíváme se „dovnitř" systémů pro řešení podmínek

- **návrh filtračních algoritmů**
  - vlastní „globální" podmínky
  - použití reifikace
  - reifikovatelné podmínky
- **návrh prohledávacích algoritmů**
  - prohledávací strategie
  - neúplná prohledávání
  - optimalizační problémy

Globální podmínky

- **Inicializace podmínky**
  - fd_global(**:Constraint**, +**State**, +**Susp**)
    - Constraint – pojmenování vlastní podmínky
    - State – počáteční globální data pro podmínku
    - Susp – seznam popisující, kdy se podmínka volá
      - dom(**X**), min(**X**), max(**X**), minmax(**X**), val(**X**)
- **Definice podmínky**
  - clpfd:dispatch_global(**+Constraint**, **+State0**, **-State**, **-Actions**)
    - filtrační algoritmus, který doporučí, jak zmenšit domény
      - exit, fail, X = V, X in R, X in_set S, call(Goal)

Příklad filtru

Jak zajistit, že v seznamu proměnných je každé proměnné přiřazena různá hodnota? **all-diff**

Myšlenka: Pokud do nějaké proměnné přiřadíme hodnotu (doména se stane jednoprvková), potom tuto hodnotu odstraníme z domén ostatních proměnných.

all_different(List)

```
all_diff(List):-
   start_all_diff(List,List).
start_all_diff([],_).
start_all_diff([H|T],List):-
   fd_global(all_diff(H,T,List),no_state,[val(H)]),
   start_all_diff(T,List).

:-multifile clpfd:dispatch_global/4.
clpfd:dispatch_global(all_diff(X,Pointer,List),S,S,Actions):-
   (ground(X) -> % value has been assigned to X
        filter_diff(List,X,Pointer, Actions)
   ;
        Actions = []
   ).

filter_diff([],_X,_Pointer, [exit]).
filter_diff([Y|T],X,Pointer, Actions):-
   (T==Pointer -> % identical objects
        Actions = RestActions
   ;
        fd_set(Y,SetY),
        fdset_del_element(SetY,X,NewSetY),
        Actions = [Y in_set NewSetY | RestActions]
   ),!,
   filter_diff(T,X,Pointer, RestActions).
```


Diff vs. all-diff

All-diff nad N proměnnými může být také popsán pomocí N.(N-1)/2 podmínek diff.

Který přístup je lepší?

- **Síla propagace**
  - Oba modely z domén odstraňují stejné hodnoty.
  - `all-distinct` realizuje silnější propagaci založenou na párování.
- **Časová efektivita**
  - all-diff je rychlejší než množina podmínek diff

Příklad:
doplnění částečného Latinského čtverce řádu 20 s 8 před-vyplněnými buňkami

- all-diff 0.68s, diff 1.43s

Latinský čtverec řádu N je matice rozměru N×N vyplněná hodnotami {1,...,N} tak, že hodnoty v každém řádku a sloupci jsou navzájem různé. **Částečný Latinský čtverec** má některé buňky před-vyplněné.

4	1	3	2
1	4	2	3
2	3	4	1
3	2	1	4

Reifikace

- umožňuje nastavit/zjistit pravdivostní hodnotu podmínky
- realizováno pomocí logické podmínky pro ekvivalenci

  `Constraint #<=> B`

  Příklad:
  - `X#>5 #<=> B` //domény X a B zůstanou stejné
  - po přidání `X#<3` bude platit X in inf..2 a B=0
  - po přidání `X#>8` bude platit X in 9..sup a B=1
  - po nastavení `B=1` bude platit X in 6..sup

`Constraint` musí být „reifikovatelný", tj. musí podporovat spojování pomocí logických operací (běžné aritmetické podmínky jsou reifikovatelné, globální podmínky zpravidla nejsou reifikovatelné).

Podmínka "exactly"

`exactly(N,List,X)`

`N` je FD proměnná, **`List`** je seznam FD proměnných a **`X`** je FD proměnná

Sémantika:

přesně **`N`** proměnných ze seznamu **`List`** se rovná **`X`**

Implementace pomocí reifikace:

```
exactly(0, [], _X).
exactly(N, [Y|L], X) :-
  X #= Y #<=> B,
  N #= M+B,
  exactly(M, L, X).
```


Filter "exactly"

`exactly(N,List,X)`

Podobně jako u **`all-diff`**, můžeme pro **`exactly`** definovat specializovaný filtrační algoritmus.

- **Základní myšlenka:**
  - Undecided ← {Y∈List | Y=X is not decided yet}
  - NumX ← |{Y∈List | Y=X}|
  - max(N)=NumX ⇒ N=NumX & ∀Y∈Undecided Y≠X
  - max(N)<NumX ⇒ fail
  - max(N)>NumX
    - MaxNumX ← NumX+|Undecided|
    - MaxNumX=min(N) ⇒ N=MaxNumX & ∀Y∈Undecided Y=X
    - MaxNumX<min(N) ⇒ fail
    - MaxNumX>min(N) ⇒ N in NumX..MaxNumX
      - NumX<min(N) ⇒ X in domain_union(Undecided)

Primitivní podmínky

- definice podmínek pomocí "reaktivních" pravidel tzv. **Indexicals**
- Odděleně se definuje propagace pozitivní a negativní podmínky a jejich splnění
  - Head +: Indexicals.
  - Head -: Indexicals.
  - Head +? Indexical.
  - Head -? Indexical.
- Tyto podmínky lze potom používat v reifikované podobě (například jako součást logických výrazů.

Primitivní podmínky
příklad

- **Propagace okrajů**

```
plus(X,Y,T) +:
      X in min(T) - max(Y) .. max(T) - min(Y),
      Y in min(T) - max(X) .. max(T) - min(X),
      T in min(X) + min(Y) .. max(X) + max(Y).
```

- "plná" propagace (**hranová konzistence**)

```
plusd(X,Y,T) +:
      X in dom(T) - dom(Y),
      Y in dom(T) - dom(X),
      T in dom(X) + dom(Y).
```

Indexicals

- Obecně mají tvar **X in R**.
  - Práce s doménami
    - dom(X), {T1,…,Tn}, T1..T2
    - R1 /\ R2, R1 \/ R2, \R1, R1+R2, R1-R2
    - …
  - Práce s termy
    - min(X), max(X), card(X)
    - X (čeká, dokud X není přiřazené), I (celé číslo), inf, sup
    - T1+T1, T1+T2, T1*T2, T1 mod T2, T1 rem T2
    - …

Primitivní podmínky

komplexní příklad

```
'x\\=y'(X,Y) +:
        X in \{Y},
        Y in \{X}.
'x\\=y'(X,Y) -:
        X in dom(Y),
        Y in dom(X).
'x\\=y'(X,Y) +?
        X in \dom(Y).
'x\\=y'(X,Y) -?
        X in {Y}.
```

propagace pozitivní části podmínky

propagace negativní části podmínky

ověření splnění podmínky

ověření nesplnění podmínky