Politechnika Poznańska

Adam Meissner

Adam.Meissner@put.poznan.pl

http://www.man.poznan.pl/~ameis

PROGRAMOWANIE WIELOPARADYGMATOWE

Wykład 2

Programowanie w paradygmatach CP(FD) i CLP(FD)

Literatura

[1] Apt K.R., *Principles of Constraint Programming*, Cambridge Univ. Press, 2003.

[2] Niederliński A., *Programowanie w logice z ograniczeniami. Łagodne wprowadzenie dla platformy ECLiPSe*, Wyd. Pracowni Komputerowej Jacka Skalmierskiego, Gliwice, 2014

[3] Van Roy P., Haridi S., *Concepts, Techniques, and Models of Computer Programming*, The MIT Press, Cambridge, USA, 2004.

[4] Smolka G., Schulte Ch., *Finite Domain Constraint Programming in Oz. A Tutorial*, Mozart Consortium, 2008.

Paradygmaty CP(FD) i CLP(FD) (1)

Problem rozwiązań symetrycznych

- do zbioru *rozwiązań symetrycznych* (ang. *symmetric solutions*) problemu CSP należą rozwiązania "te same" z dokładnością do pewnego izomorfizmu
- eliminowanie rozwiązań symetrycznych może wydatnie zwiększyć efektywność poszukiwania wszystkich rozwiązań problemu CSP i stanowi jedno z podstawowych zagadnień rozpatrywanych w ramach teorii programowania z ograniczeniami
- jedną z podstawowych technik eliminowania rozwiązań symetrycznych jest wprowadzanie porządku do wartościowań poszczególnych zmiennych

Przykład 2.1 (wg [4])

Sformułowanie problemu

W sklepie zakupiono 4 różne produkty; kasjer podał ich cenę łączną - 7,11 zł (711 gr), zaraz jednak poprawił się wyjaśniając, że zamiast zsumować poszczególne kwoty, wykonał ich mnożenie. Jednakże, po zsumowaniu, cena łączna nie zmieniła się. Podać ceny poszczególnych produktów.

Paradygmaty CP(FD) i CLP(FD) (2)

Przykład 2.1 (c.d.)

Implementacja

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

grocery(Vars) :-
   Vars = [A,B,C,D], S is 711,
   Vars ins 1..S,
   A * B * C * D #= S * 100 * 100 * 100,
   A + B + C + D #= S,
%  A #= 79 * _,
   A #=< B,
   B #=< C,
   C #=< D,
   labeling([ff],Vars).
```

Paradygmaty CP(FD) i CLP(FD) (3)

Nakładanie więzów za pomocą podprogramów

Do nakładania ograniczeń, oprócz bezpośredniego stosowania propagatorów bibliotecznych, można wykorzystywać "własne" podprogramy.

Przykład 2.2 (wg [4])

Sformułowanie problemu

1. Maria i Klara są matkami 2 rodzin
2. W każdej rodzinie jest 3 synów i 3 córki
3. Różnica wieku między dziećmi nie jest w żadnym przypadku mniejsza niż 1 rok
4. Wszystkie dzieci mają mniej niż 10 lat
5. W rodzinie Klary właśnie urodziła się córka
6. Najmłodszym dzieckiem Marii jest dziewczynka.
7. W obu rodzinach suma wieku chłopców jest równa sumie wieku dziewczynek; równość zachodzi także między sumami kwadratów wieku chłopców i dziewczynek
8. Suma wieku wszystkich dzieci wynosi 60

Ile lat mają poszczególne dzieci w każdej z rodzin?

Elementy modelu problemu

Rodzinę reprezentuje się za pomocą termu `Name(boys:[B1,B2,B3], girls:[G1,G2,G3])`. Zmienna `Name` reprezentuje imię matki; elementy obu list są uszeregowane malejąco.

Paradygmaty CP(FD) i CLP(FD) (4)

Przykład 2.2 (wg [4], c.d.)
Implementacja

```
family([Maria,Clara]) :-
   is_family(maria,Maria),
   is_family(clara,Clara),
   children_ages(Maria,girls,MGirls),
   children_ages(Maria,boys,MBoys),
   children_ages(Clara,girls,CGirls),
   children_ages(Clara,boys,CBoys),
   element(3,MGirls,MGYoungestAge),
   element(3,CGirls,CGYoungestAge),
   append([CGirls,CBoys,MGirls,MBoys],Ages),
   every_boy_elder(MBoys,MGYoungestAge),
   CGYoungestAge is 0,
   sum(Ages,#=,60),
   labeling([ff],Ages).

is_family(Name,Term) :-
   Coeffs = [1,1,1,-1,-1,-1],
   age_list(BAgeList),
   age_list(GAgeList),
   Term =.. [Name,boys:BAgeList,girls:GAgeList],
   append(BAgeList,GAgeList,Ages),
   length(Ages,AgeLen),
   length(SqrAges,AgeLen),
   SqrAges ins inf..sup,
   maplist(sqr,Ages,SqrAges),
   all_different(Ages),
   scalar_product(Coeffs,Ages,#=,0),
   scalar_product(Coeffs,SqrAges,#=,0).

age_list(AgeList) :-
   AgeList = [A1,A2,A3],
   AgeList ins 0..9,
   A1 #> A2, A2 #> A3.
```

Paradygmaty CP(FD) i CLP(FD) (5)

Przykład 2.2 (wg [4], c.d.)
Implementacja

```
sqr(N,SqrN) :- SqrN #= N * N.

children_ages(Mother,boys,List) :-
   Mother =.. [_,boys:List,_].
children_ages(Mother,girls,List) :-
   Mother =.. [_,_,girls:List].

every_boy_elder([],_) :- !.
every_boy_elder([B|Bs],Age) :-
   B #> Age,
   every_boy_elder(Bs,Age).
```

Paradygmaty CP(FD) i CLP(FD) (6)

Przykład 2.3 (wg [4], zagadka Einsteina)

Sformułowanie problemu

W 5 domach położonych wzdłuż (po jednej stronie) ulicy mieszka 5 mężczyzn różnej narodowości. Każdy z nich ma inny zawód, inny ulubiony napój i inne ulubione zwierzę. Każdy dom ma inny kolor. Znane są przy tym następujące fakty.

1. Anglik mieszka w domu koloru czerwonego
2. Hiszpan ma psa
3. Japończyk jest malarzem
4. Włoch lubi herbatę
5. Norweg mieszka w pierwszym domu
6. Właściciel zielonego domu lubi kawę
7. Zielony dom następuje po domu białym
8. Rzeźbiarz hoduje ślimaki
9. Dyplomata mieszka w domu koloru żółtego
10. Właściciel trzeciego domu lubi mleko
11. Dom Norwega sąsiaduje z domem niebieskim
12. Skrzypek lubi sok
13. Lis żyje przy domu, który jest obok domu lekarza
14. Koń żyje przy domu obok domu dyplomaty
15. Zebra żyje przy domu koloru białego
16. Jeden z mieszkańców lubi wodę

Kto gdzie mieszka?

Paradygmaty CP(FD) i CLP(FD) (7)

Przykład 2.3 (wg [4], c.d.)

Elementy modelu problemu

W rozpatrywanym problemie występuje 5 cech (*narodowość*, *zawód*, *kolor domu*, *ulubione zwierzę*, *ulubiony napój*) z których każda może przyjąć 5 wartości. Z każdą wartością cechy wiąże się zmienną oznaczającą numer domu. Wykorzystuje się trzy predykaty służące do nakładania więzów: `partition(Gs,_,Ns)` postulujący, że cechy występujące na liście `Gs` muszą przysługiwać parami różnym domom o numerach z listy `Ns`; predykat `props_rel(A,B,R,_)` stwierdzający, że między numerami domów odpowiadających cechom `A` i `B` zachodzi relacja `R`, a także predykat `adjacent(A,B,_)` stwierdzającą, że cechy `A` oraz `B` przysługują domom sąsiadującym ze sobą.

Implementacja

```
partition(Group, PGroup,Numbers) :-
   length(Numbers,5),
   Numbers ins 1..5,
   all_different(Numbers),
   partition1(Group,Numbers,PGroup).

partition1([],[],[]).
partition1([P|Ps],[N|Ns],[P:N|PNs]) :-
   partition1(Ps,Ns,PNs).

props_rel(Prop1,Prop2,Rel,Props) :-
   member(Prop1:X,Props),
   member(Prop2:Y,Props),
   Goal =.. [Rel,X,Y],
   call(Goal).
```

Paradygmaty CP(FD) i CLP(FD) (8)

Przykład 2.3 (wg [4], c.d.)

Elementy modelu problemu

```
adjacent(Prop1,Prop2,Props) :-
   member(Prop1:X,Props),
   member(Prop2:Y,Props),
   abs(X-Y) #= 1.

zebra(Props) :-
   Groups = [[english,spanish,japanese,italian,
              norvegian],
             [green,red,yellow,blue,white],
             [painter,diplomat,violinist,doctor,
              sculptor],
             [dog,zebra,fox,snails,horse],
             [juice,water,tea,coffee,milk]],
   maplist(partition,Groups,PropGroups,GroupVars),
   append(PropGroups,Props),
   append(GroupVars,Vars),
   props_rel(english,red,#=,Props),
   props_rel(spanish,dog,#=,Props),
   props_rel(japanese,painter,#=,Props),
   props_rel(italian,tea,#=,Props),
   member(norvegian:N1,Props), N1 #= 1,
   props_rel(green,coffee,#=,Props),
   props_rel(green,white,#>,Props),
   props_rel(sculptor,snails,#=,Props),
   props_rel(diplomat,yellow,#=,Props),
   member(milk:N2,Props), N2 #= 3,
   adjacent(norvegian,blue,Props),
   props_rel(violinist,juice,#=,Props),
   adjacent(fox,doctor,Props),
   adjacent(horse,diplomat,Props),
   props_rel(zebra,white,#=,Props),
   labeling([ff],Vars).
```

Paradygmaty CP(FD) i CLP(FD) (9)

Przykład 2.4 (wg [4], problem *N*-hetmanów)

Sformułowanie problemu

Na szachownicy o wymiarze N × N należy rozmieścić N hetmanów w taki sposób, aby żadne dwa hetmany nie szachowały się wzajemnie.

Elementy modelu problemu

Ciąg hetmanów reprezentuje się jako *N*-elementową listę liczb, której element *R*(*i*) odpowiada hetmanowi w *i*-tej kolumnie i w *R*(*i*)-tym wierszu dla *i* = 1,… , *n*. Warunek, aby dowolne dwa hetmany, tj. *R*(*i*) oraz *R*(*j*) nie leżały na jednej przekątnej wyraża się jako

$$R(i) + (j - i) \leq R(j) \text{ oraz } R(i) - (j - i) \leq R(j) \text{ dla } 1 \leq i < j \leq n$$

lub równoważnie, aby ciągi

$$R(1) - 1, \ldots, R(N) - N \text{ oraz } R(1) + 1, \ldots, R(N) + N$$

były różnowartościowe

Implementacja

```
fun {Queens N}
   proc {$ Row}
      L1N  = {MakeTuple c N}
      LM1N = {MakeTuple c N}
   in
      Row = {FD.tuple queens N 1#N}
      {For 1 N 1 proc {$ I} L1N.I=I LM1N.I=~I end}
      {FD.distinct Row}
      {FD.distinctOffset Row L1N}
      {FD.distinctOffset Row LM1N}
      {FD.distribute
                generic(order:size value:mid) Row}
   end
end
```

Paradygmaty CP(FD) i CLP(FD) (10)

Przykład 2.5 (wg [4])

Sformułowanie problemu

Dana jest pewna liczba monet o różnych nominałach. Niech a_i oznacza liczbę dostępnych monet o nominale d_i dla i = 1, …, n. Wyznaczyć minimalną liczbę monet niezbędną do wypłacenia kwoty A.

Elementy modelu problemu

Informację o dostępnych monetach reprezentuje lista `Coins` postaci $[d_1:a_1, \ldots, d_n:a_n]$. Rozwiązanie problemu reprezentuje lista $[C_1, \ldots, C_n]$ taka, że dla $i = 1, \ldots, n$ element C_i jest liczbą monet o nominale d_i, przy czym $0 \le C_i \le a_i$.

Implementacja

```
change(Coins,Amount,Changes) :-
   maplist(split_pair,Coins,Denoms,Avails),
   maplist(avail_constr,Avails,Changes),
   scalar_product(Denoms,Changes,#=,Amount),
   sum(Changes,#=,CoinsNo),
   labeling([min(CoinsNo)],Changes).

split_pair(D:A,D,A).

avail_constr(Avail,Change) :- Change in 0..Avail.
```