Politechnika Poznańska

Adam Meissner

Adam.Meissner@put.poznan.pl

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

PROGRAMOWANIE WIELOPARADYGMATOWE

Wykład 4

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)

Przykład 4.1 (wg [4], program konferencji)

Sformułowanie problemu

Ułożyć program (plan) konferencji w formie ciągu paneli. Konferencja składa się z 11 sesji a w każdym panelu może toczyć się równolegle do 3 sesji. Ponadto, obowiązują następujące ograniczenia.

Sesja 4 musi odbyć się przed sesja 11.

Sesja 5 musi odbyć się przed sesja 10.

Sesja 6 musi odbyć się przed sesja 11.

Sesja 1 nie może toczyć się równolegle z sesjami 2, 3, 5, 7, 8 oraz 10

Sesja 2 nie może toczyć się równolegle z sesjami 3, 4, 7, 8, 9 oraz 11

Sesja 3 nie może toczyć się równolegle z sesjami 5, 6 oraz 8

Sesja 4 nie może toczyć się równolegle z sesjami 6, 8 oraz 10.

Sesja 6 nie może toczyć się równolegle z sesjami 7 oraz 10

Sesja 7 nie może toczyć się równolegle z sesjami 8 oraz 9

Sesja 8 nie może toczyć się równolegle z sesją 10

Znaleźć rozwiązanie o minimalnej liczbie paneli

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

Przykład 4.1 (wg [4], cd.)

Elementy modelu problemu

Do reprezentowania liczby paneli służy zmienna `NbSlots in 4..11`. Rozwiązanie ma postać termu `plan(`p_1`,...,`p_{11}`)` takiej, że element p_i dla $i = 1, ..., n$ oznacza panel, w którym odbywa się *i*-ta sesja; dla każdego *i*, p_i `n 1..NbSlots`.

Implementacja

```
for_at_most(Slot,NbSlots,NbParSessions,Plan) :-
  Slot =< NbSlots ->
     at_most(NbParSessions,Plan,Slot),
     Slot1 is Slot + 1,
     for_at_most(Slot1,NbSlots,NbParSessions,Plan)
  ;
  true.

at_most(D, Dv, I) :-
   maplist(eq_b(I), Dv, Bs),
   sum(Bs, #=<, D).

eq_b(X, Y, B) :- X #= Y #<==> B.

disjoint(Plan,X,Y,_) :-
   nth1(X,Plan,Px),
   nth1(Y,Plan,Py),
   Px #\= Py.
```

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

Przykład 4.1 (wg [4], c.d.)

Implementacja

```
conference(Plan) :-
   Data = [nbSessions(11),
           nbParSessions(3),
           constraints(
               [before(4,11),
                before(5,10),
                before(6,11),
                disjoint(1, [2,3,5,7,8,10]),
                disjoint(2, [3,4,7,8,9,11]),
                disjoint(3, [5,6,8]),
                disjoint(4, [6,8,10]),
                disjoint(6, [7,10]),
                disjoint(7, [8,9]),
                disjoint(8, [10])])],
   member(nbSessions(NbSessions),Data),
   member(nbParSessions(NbParSessions),Data),
   member(constraints(Constraints),Data),
   MinNbSlots is ceiling(NbSessions / NbParSessions),
   NbSlots in MinNbSlots .. NbSessions,
   labeling([leftmost],[NbSlots]),
   length(Plan,NbSessions),
   Plan ins 1 .. NbSlots,
   for_at_most(1,NbSlots,NbParSessions,Plan),
   maplist(constraint(Plan),Constraints),
   labeling([ff],Plan).

constraint(Plan,before(X,Y)) :-
   nth1(X,Plan,Px),
   nth1(Y,Plan,Py),
   Px #< Py.
constraint(Plan,disjoint(X,Ys)) :-
               maplist(disjoint(Plan,X),Ys).
```

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

Przykład 4.2 (wg [4], autoreferencyjny test wyboru)

Sformułowanie problemu

Dany jest test wyboru składający się z 10-ciu pytań. Na każde z nich formułuje się 5 możliwych odpowiedzi (numerowanych od *a* do *e*), z których dokładnie 1 jest poprawna. Znaleźć odpowiedzi poprawne, spełniające wszystkie warunki podane w pytaniach.

1. Pierwszym pytaniem, na które odpowiedź brzmi *b* jest pytanie: (a) 2, (b) 3, (c) 4, (d) 5, (e) 6.
2. Jedyną parą sąsiadujących ze sobą pytań o identycznym numerze odpowiedzi są pytania: (a) 2-3, (b) 3-4, (c) 4-5, (d) 5-6, (e) 6-7.
3. Odpowiedź na niniejsze pytanie jest taka sama jak odpowiedź na pytanie: (a) 1, (b) 2, (c) 4, (d) 7, (e) 6.
4. Liczba pytań, na które odpowiedź brzmi *a* wynosi: (a) 0, (b) 1, (c) 2, (d) 3, (e) 4.
5. Odpowiedź na niniejsze pytanie jest taka sama jak odpowiedź na pytanie: (a) 10, (b) 9, (c) 8, (d) 7, (e) 6.
6. Liczba pytań, na które odpowiedź brzmi *a* jest równa liczbie pytań, na które odpowiedź brzmi: (a) b, (b) c, (c) d, (d) e, (e) żadnej z wymienionych.
7. Alfabetyczna odległość odpowiedzi na pytanie niniejsze i następne wynosi: (a) 4, (b) 3, (c) 2, (d) 1, (e) 0.
8. Liczba pytań o odpowiedziach oznaczonych samogłoskami wynosi: (a) 2, (b) 3, (c) 4, (d) 5, (e) 6.
9. Liczba odpowiedzi oznaczonych spółgłoskami jest: (a) liczbą pierwszą, (b) silnią pewnej liczby, (c) kwadratem, (d) sześcianem , (e) jest podzielna całkowicie przez 5.
10. Odpowiedzią na niniejsze pytanie jest: (a) a, (b) b, (c) c, (d) d, (e) e.

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

Przykład 4.2 (wg [4], c.d.)

Uwaga, problem ma tylko 1 następujące rozwiązanie:

1:c 2:d 3:e 4:b 5:e 6:e 7:d 8:c 9:b 10:a

Elementy modelu problemu

W modelu wykorzystuje się zmienne A_i, B_i, …, E_i o dziedzinach 0#1 dla $i \in$ 1#10. Każda ze zmiennych ma wartość 1 wtw odpowiedzią na i-te pytanie jest litera będąca nazwą zmiennej. Wiadomo, że $A_i + B_i + \ldots + E_i = 1$. Dodatkowo wprowadza się zmienne Q_i dla o dziedzinach 1#5 dla $i \in$ 1#10, takie że

$Q_i = 1 \leftrightarrow A_i = 1$ $Q_i = 2 \leftrightarrow B_i = 1$ $Q_i = 3 \leftrightarrow C_i = 1$

$Q_i = 4 \leftrightarrow D_i = 1$ $Q_i = 5 \leftrightarrow E_i = 1$

Ograniczenia reprezentujące warunek 1

$A_1 = B_2$

$B_1 = (B_3 \wedge (B_2 = 0))$

$C_1 = (B_4 \wedge (B_2 + B_3 = 0))$

$D_1 = (B_5 \wedge (B_2 + B_3 + B_4 = 0))$

$E_1 = (B_6 \wedge (B_2 + B_3 + B_4 + B_5 = 0))$

Ograniczenia reprezentujące warunek 2

$Q_1 \neq Q_2$, $Q_7 \neq Q_8$, $Q_8 \neq Q_9$, $Q_9 \neq Q_{10}$,

$A_2 = (Q_2 = Q_3)$, $B_2 = (Q_3 = Q_4)$, $C_2 = (Q_4 = Q_5)$,

$D_2 = (Q_5 = Q_6)$, $E_2 = (Q_6 = Q_7)$

Ograniczenia reprezentujące warunek 3

$A_3 = (Q_1 = Q_3)$, $B_3 = (Q_2 = Q_3)$, $C_3 = (Q_3 = Q_4)$,

$D_3 = (Q_7 = Q_3)$, $E_3 = (Q_6 = Q_3)$

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

Przykład 4.2 (wg [4], c.d.)

Ograniczenia reprezentujące warunek 4

element(Q_4, (0, 1, 2, 3, 4)) = $A_1 + A_2 + \ldots + A_{10}$

Ograniczenia reprezentujące warunek 9

$S = (B_1 + C_1 + D_1) + \ldots + (B_{10} + C_{10} + D_{10})$

$A_9 = (S \in \{2, 3, 5, 7\})$

$B_9 = (S \in \{1, 2, 6\})$

$C_9 = (S \in \{0, 1, 4, 9\})$

$D_9 = (S \in \{0, 1, 8\})$

$E_9 = (S \in \{0, 5, 10\})$

Implementacja (Oz)

```
proc {SRAT Q}
   proc {Vector V}
      {FD.tuple v 10 0#1 V}
   end
   proc {Sum V S}
      {FD.decl S}
      {FD.sum V '=:' S}
   end
   proc {Assert I [U V W X Y]}
      A.I=U  B.I=V  C.I=W  D.I=X  E.I=Y
   end
   A = {Vector} B = {Vector} C = {Vector}  D = {Vector} E = {Vector}
   SumA ={Sum A}  SumB = {Sum B}  SumC = {Sum C}  SumD = {Sum D}
   SumE = {Sum E} SumAE = {Sum [SumA SumE]} SumBCD = {Sum [SumB SumC SumD]}
in
   {FD.tuple q 10 1#5 Q}
   {For 1 10 1
    proc {$ I} {Assert I [Q.I=:1  Q.I=:2  Q.I=:3  Q.I=:4  Q.I=:5]} end}
   %% 1
   {Assert 1 [ B.2
             {FD.conj B.3 (B.2=:0)}
             {FD.conj B.4 (B.2+B.3=:0)}
             {FD.conj B.5 (B.2+B.3+B.4=:0)}
             {FD.conj B.6 (B.2+B.3+B.4+B.5=:0)} ]}
```

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

Przykład 4.2 (wg [4], c.d.)

```
   %% 2
   {Assert 2 [Q.2=:Q.3  Q.3=:Q.4  Q.4=:Q.5  Q.5=:Q.6  Q.6=:Q.7]}
   Q.1\=:Q.2  Q.7\=:Q.8  Q.8\=:Q.9  Q.9\=:Q.10
   %% 3
   {Assert 3 [Q.1=:Q.3  Q.2=:Q.3  Q.4=:Q.3  Q.7=:Q.3  Q.6=:Q.3]}
   %% 4
   {FD.element Q.4 [0 1 2 3 4] SumA}
   %% 5
   {Assert 5 [Q.10=:Q.5  Q.9=:Q.5  Q.8=:Q.5  Q.7=:Q.5  Q.6=:Q.5]}
   %% 6
   {Assert 6 [SumA=:SumB  SumA=:SumC  SumA=:SumD  SumA=:SumE  _]}
   %% 7
   {FD.element Q.7 [4 3 2 1 0] {FD.decl}={FD.distance Q.7 Q.8 '=:'}}
   %% 8
   {FD.element Q.8 [2 3 4 5 6] SumAE}
   %% 9
   {Assert 9 [SumBCD::[2 3 5 7]  SumBCD::[1 2 6]     SumBCD::[0 1 4 9]
              SumBCD::[0 1 8]   SumBCD::[0 5 10] ]}
   %% 10
   skip
   {FD.distribute ff Q}
end
```

Implementacja (SWI-Prolog)

```
srat(Q) :-
   srat_vector(A), srat_vector(B), srat_vector(C),
   srat_vector(D), srat_vector(E), srat_sum(A,SumA),
   srat_sum(B,SumB), srat_sum(C,SumC),
   srat_sum(D,SumD), srat_sum(E,SumE),
   srat_sum([SumA,SumE],SumAE),
   srat_sum([SumB,SumC,SumD],SumBCD),
   Q = [Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9, Q10],
   Q ins 1..5,
   srat_for_assert(Q,[A,B,C,D,E],10),
   LstsAE = [A,B,C,D,E],
   B = [_,B2,B3,B4,B5,B6|_],                                  % Q-1
   srat_assert(1, B2,
                (B3 #/\ (B2 #= 0)),
                (B4 #/\ (B2+B3 #= 0)),
                (B5 #/\ (B2+B3+B4#=0)),
                (B6 #/\ (B2+B3+B4+B5#=0)),LstsAE),
```

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

Przykład 4.2 (wg [4], c.d.)

```
srat_assert(2, Q2#=Q3, Q3#=Q4, Q4#=Q5, Q5#=Q6, Q6#=Q7, LstsAE),     % Q-2
Q1 #\= Q2, Q7 #\= Q8, Q8 #\= Q9, Q9 #\= Q10,
srat_assert(3, Q1#=Q3, Q2#=Q3, Q4#=Q3, Q7#=Q3, Q6#=Q3, LstsAE),    % Q-3
element(Q4,[0,1,2,3,4],SumA),                                       % Q-4
srat_assert(5, Q10#=Q5, Q9#=Q5, Q8#=Q5, Q7#=Q5, Q6#=Q5, LstsAE),    % Q-5
srat_assert(6, SumA#=SumB, SumA#=SumC, SumA#=SumD, SumA#=SumE,_, LstsAE),   % Q-6
Dist in inf..sup,                                                   % Q-7
abs(Q8-Q7) #= Dist,
element(Q7,[4,3,2,1,0],Dist),
element(Q8,[2,3,4,5,6],SumAE),                                      % Q-8
srat_assert(9, (SumBCD in 2\/3\/5\/7), (SumBCD in 1\/2\/6),    % Q-9
               (SumBCD in 0\/1\/4\/9), (SumBCD in 0\/1\/8),
               (SumBCD in 0\/5\/10), LstsAE),
labeling([ff],Q).

srat_vector(V) :-
   length(V,10),
   V ins 0..1.

srat_sum(V,S) :-
   S in inf..sup,
   sum(V,#=,S).

srat_assert(I,U,V,W,X,Y,LstsAE) :-
   maplist(srat_set_val(I),LstsAE,[U,V,W,X,Y]).

srat_set_val(Ind,Lst,Val) :-
   nth1(Ind,Lst,Elem),
   Elem #<==> Val.

srat_for_assert(_,_,0) :- !.
srat_for_assert(Q,LstsAE,I) :-
   nth1(I,Q,E),
   srat_assert(I,E#=1,E#=2,E#=3,E#=4,E#=5,LstsAE),
   I1 is I - 1,
   srat_for_assert(Q,LstsAE,I1).

num_to_char(N,C) :- N1 is N+96, char_code(C,N1).

main :-
   srat(Q),
   write('Answers to subsequent SRAT questions: '),
   maplist(num_to_char,Q,Q1),
   writeln(Q1).
```

Zasady dobrego programowania

1. Dobrze przeanalizuj problem i program

Aby to osiągnąć, używaj narzędzi do wizualizacji drzewa poszukiwań oraz do wyznaczania statystyk związanych z działaniem programu.

2. Dużo eksperymentuj

Dla danego programu wypróbuj różne strategie dystrybucji, różne maszyny przeszukujące, próbuj dodawać ograniczenia redundantne.

3. Używaj jak najwięcej propagacji

Minimalizuj drzewo poszukiwań, dodawaj propagatory eliminujące symetrie, dodawaj ograniczenia nadmiarowe.

4. Znajdź właściwą strategię dystrybucji

Należy pamiętać, że strategia dystrybucji determinuje postać drzewa poszukiwań. Poszukiwanie właściwej strategii (wobec braku dodatkowych przesłanek) można zaczynać od strategii *first fail*.

5. Minimalizuj liczbę zmiennych i propagatorów

Wykorzystanie pamięci w znacznym stopniu zależy od liczby propagatorów i zmiennych. Należy zauważyć, że niniejsza reguła może być w opozycji do reguły 3.