Politechnika Poznańska

Adam Meissner

Adam.Meissner@put.poznan.pl

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

PROGRAMOWANIE WIELOPARADYGMATOWE

Wykład 3

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)

Optymalizacyjne problemy CSP

- do rozpatrywanej klasy należą problemy, z którymi wiąże się konieczność znalezienia rozwiązania najlepszego w zbiorze *R* wszystkich rozwiązań problemu
- w szczególności, optymalizacja może polegać na minimalizowaniu funkcji ceny określonej na zbiorze *R*
- *w niektórych (prostych) przypadkach minimalizację funkcji ceny można uzyskać poprzez zastosowanie 2-wymiarowej strategii dystrybucji*

Przykład 3.1 (wg [4], kolorowanie mapy)

Sformułowanie problemu

Dana jest polityczna mapa Europy Zachodniej, na której widnieje Holandia, Belgia, Francja, Hiszpania, Portugalia, Niemcy, Luksemburg, Szwajcaria, Austria i Włochy. Używając jak najmniejszej liczby kolorów pomalować mapę tak, aby żadne dwa kraje graniczące ze sobą nie miały tej samej barwy.

Elementy modelu problemu

Rozwiązanie problemu reprezentuje lista $[r_1 : c_1 \ , \ldots , \ r_n : c_n)$ taka, że dla $i = 1, \ldots, n$ kraj o nazwie r_i ma na mapie kolor wyrażony przez liczbę c_i ; dziedziną zmiennej c_i jest zbiór `1..NbColors`.

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

Przykład 3.1 (wg [4], c.d.)

Implementacja

```
map_color(Result):-
   Data =
      [[austria, [italy,switzerland,germany]],
       [belgium, [france,netherlands,germany,
                  luxemburg]],
       [france, [spain,luxemburg,italy]],
       [germany, [austria,france,luxemburg,
                  netherlands]],
       [italy, []],
       [luxemburg, []],
       [netherlands, []],
       [portugal, []],
       [spain, [portugal]],
       [switzerland, [italy,france,germany,
                      austria]]],
   maplist(country,Data,Countries),
   length(Countries,NbCountries),
   length(CountryColors,NbCountries),
   NbColors in 1..NbCountries,
   labeling([ff],[NbColors]),
   CountryColors ins 1..NbColors,
   maplist(color(Countries,CountryColors),Data),
   labeling([ff],CountryColors),
   maplist(result,Countries,CountryColors,Result).

country([Country,_],Country).

color(Cnts,CntCols,[Cnt,Neighs]) :-
   country_color(Cnts,CntCols,Cnt,CntCol),
   maplist(color1(Cnts,CntCols,CntCol),Neighs).

result(Cnt,CntCol,Cnt:CntCol).
```

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

Przykład 3.1 (wg [4], c.d.)

Implementacja

```
color1(Cnts,CntCols,CntCol,Neigh) :-
   country_color(Cnts,CntCols,Neigh,NeighCol),
   CntCol #\= NeighCol.

country_color([Cnt|_],[Col|_],Cnt,Col).
country_color([_|Cnts],[_|Cols],Cnt,Col) :-
   country_color(Cnts,Cols,Cnt,Col).
```

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

Ograniczenia nadmiarowe (redundantne)

Niech *P* oraz *S* będą zbiorami ograniczeń, zbiór S jest redundantny względem *P*, jeżeli *S* jest konsekwencją logiczną *P*; przykładowo, zbiór ograniczeń {X<:10} jest redundantny względem zbioru {X<:5}.

Wprowadzenie odpowiednich ograniczeń redundantnych może zmniejszyć drzewo poszukiwań, jak również zredukować liczbę kroków propagacji wykonywanych w przestrzeniach obliczeń.

Przykład 3.2 (wg [4], ułamki)

Sformułowanie problemu

Znaleźć przypisanie cyfr różnych od 0 do liter występujących w poniższym wyrażeniu tak, aby uzyskać poprawne wyrażenie arytmetyczne

$$(A / BC) + (D / EF) + (G/HI) = 1$$

Elementy modelu problemu

W celu usunięcia rozwiązań symetrycznych, wprowadza się następujący porządek na wartościowania zmiennych:

$$(A / BC) \geq (D / EF) \geq (G/HI).$$

Na tej podstawie można sformułować następujące ograniczenia redundantne:

$$3 * (A / BC) \geq 1 \text{ oraz } 3 * (G/HI) \leq 1$$

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

Przykład 3.2 (wg [4], cd.)

Implementacja

```
proc {Fraction Root}
   sol(a:A b:B c:C d:D e:E f:F g:G h:H i:I) = Root
   BC = {FD.decl}
   EF = {FD.decl}
   HI = {FD.decl}
in
   Root ::: 1#9
   {FD.distinct Root}
   BC =: 10*B + C
   EF =: 10*E + F
   HI =: 10*H + I
   A*EF*HI + D*BC*HI + G*BC*EF =: BC*EF*HI

   A*EF >=: D*BC
   D*HI >=: G*EF

   3*A >=: BC
   3*G =<: HI
   {FD.distribute split Root}
end
```

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

Przykład 3.3 (wg [4], liczby Pitagorasa)

Sformułowanie problemu

Wyznaczyć liczbę trójek (A, B, C), takich że $A \leq B \leq C \leq 1000$ i $A^2 + B^2 = C^2$.

Elementy modelu problemu

Do reprezentowania liczb Pitagorasa służą zmienne A, B, C; dziedziną początkową każdej z nich jest zbiór 1#1000. Z ograniczenia $A^2 + B^2 = C^2$ można wyprowadzić następujące ograniczenie redundantne:

$$2 * B^2 \geq C^2$$

Implementacja

```
proc {Pythagoras Root}
   [A B C] = Root
   AA BB CC
in
   Root ::: 1#1000
   AA = {FD.times A A}
   BB = {FD.times B B}
   CC = {FD.times C C}
   AA + BB =: CC
   A =<: B
   B =<: C

   % ograniczenie nadmiarowe
   2*BB >=: CC

   {FD.distribute ff Root}
end
```

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

Przykład 3.4 (wg [4], kwadraty magiczne)

Sformułowanie problemu

Kwadratem magicznym rzędu N nazywa się macierz $N \times N$ zawierającą liczby naturalne z przedziału [1, N^2], ułożone tak aby suma elementów w każdym wierszu i w każdej kolumnie oraz na obu przekątnych gł. była taka sama (jest to tzw. *suma magiczna*). Przykładowo, poniższa macierz jest kwadratem magicznym rzędu 3.

2	7	6
9	5	1
4	3	8

Elementy modelu problemu

Do reprezentowania elementu (*i*, *j*) kwadratu używa się zmiennej $F_{i,j}$, a zmienna *S* oznacza sumę magiczną. Obowiązują następujące ograniczenia.

$F_{1,1} < F_{N,N}$, $F_{N,1} < F_{1,N}$, $F_{1,1} < F_{N,1}$ (elim. symetrii)

$0.5 * N^2 * (N^2 + 1) = S * N$ (ogr. redundantne)

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

Przykład 3.4 (wg [4], cd.)

Implementacja

```
fun {MagicSquare N}
   NN  = N*N
   L1N = {List.number 1 N 1}  % [1 2 3 ... N]
in
   proc {$ Square}
      fun {Field I J}
         Square.((I-1)*N + J)
      end
      proc {Assert F}
         %% {F 1} + {F 2} + ... + {F N} =: Sum
         {FD.sum {Map L1N F} '=:' Sum}
      end
      Sum = {FD.decl}
   in
      {FD.tuple square NN 1#NN Square}
      {FD.distinct Square}
      %% Przekątne
      {Assert fun {$ I} {Field I I} end}
      {Assert fun {$ I} {Field I N+1-I} end}
      %% Kolumny
      {For 1 N 1
       proc {$ I} {Assert fun {$ J} {Field I J} end} end}
      %% Wiersze
      {For 1 N 1
       proc {$ J} {Assert fun {$ I} {Field I J} end} end}
      %% Eliminacja symetrii
      {Field 1 1} <: {Field N N}
      {Field N 1} <: {Field 1 N}
      {Field 1 1} <: {Field N 1}
      %% Ogr.nadm.: suma pól = (liczba wierszy) * Sum
      NN*(NN+1) div 2 =: N*Sum

      {FD.distribute split Square}
   end
end
```

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

Ograniczenia reifikowane

- ograniczenia reifikowane są przeznaczone do wyrażania więzów, w których wykorzystuje się symbole logiczne, takie jak alternatywa, negacja czy implikacja
- *reifikacją ograniczenia C ze względu na zmienną x* nazywa się ograniczenie:

$$(C \leftrightarrow x = 1) \wedge x \in 0\#1$$

  i *x* nie występuje jako zmienna wolna w *C*
- semantyka operacyjna propagatora będącego reifikacją ograniczenia *C* ze względu na zmienną *x*:

  1) jeżeli ogr. $x = 1$ jest nadmiarowe w danym składzie więzów, to ogr. reifikowane redukuje się do *C;*

  2) jeżeli ogr. $x = 0$ jest nadmiarowe w danym składzie więzów, to ogr. reifikowane redukuje się do $\neg C$;

  3) jeżeli propagator dla ogr. reifikowanego jest nadmiarowy w danym składzie więzów, to do składu wprowadza się ogr. $x = 1$ i propagator zostaje usunięty;

  4) jeżeli propagator dla ogr. reifikowanego jest sprzeczny z danym składem więzów, to do składu wprowadza się ogr. $x = 0$ i propagator zostaje usunięty.

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

Ograniczenia reifikowane (c.d.)

- ograniczenie będące reifikacją ograniczenia *C* ze względu na zmienna *x* intuicyjnie wyraża to, że formuła *C* jest prawdziwa wtw gdy x ma wartość 1 a fałszywa wtw gdy x ma wartość 0
- przykładowo, poniższa formuła wyraża równoważność dwóch ograniczeń

$$X < Y \leftrightarrow X < Z$$

formułę tę można wyrazić za pomocą poniższych ograniczeń reifikowanych

```
X <: Y = B    X <: Z = B
```

Przykład 3.5 (wg [4], ustawianie do zdjęcia)

Sformułowanie problemu

Beata, Krzyś, Donald, Franek, Grześ, Maria i Paweł ustawiają się w jednym rzędzie do zdjęcia. Mają przy tym następujące oczekiwania.

1. Beata chce stać między Grzesiem a Marią
2. Krzyś chce stać między Beatą a Grzesiem
3. Franek chce stać między Marią a Donaldem
4. Paweł chce stać między Frankiem a Donaldem

Znaleźć ustawienie, spełniające możliwie jak największą liczbę oczekiwań fotografowanych osób.

Paradygmaty CP(FD) i CLP(FD) (11)

Przykład 3.5 (wg [4], cd.)

Elementy modelu problemu

Do reprezentowania położenia osoby *p* w rzędzie służy zmienna $A_P \in 1\#7$. Zmienna $S_i \in 0\#1$ ma wartość 1 wtw *i*-ta preferencja jest spełniona. Ściślej ujmując, ograniczenie wyrażające fakt, że osoba A_P chce stać obok A_Q osoby ma postać formuły

$$(|A_P - A_Q| = 1 \leftrightarrow S = 1) \wedge S \in 0\#1$$

Rozwiązanie problemu polega na znalezieniu maksymalnej wartości zmiennej *Sat*, takiej że

$$Sat = S_1 + \ldots + S_8$$

Implementacja

```
proc {Photo Root}
   Persons = [beata krzys donald franek grzes maria pawel]
   Preferences = [beata#grzes beata#maria krzys#beata
                  krzys#grzes franek#maria franek#donald
                  pawel#franek pawel#donald]
   NbPersons = {Length Persons}
   Alignment = {FD.record alignment Persons 1#NbPersons}
   Sat  = {FD.decl}
   proc {Satisfied P#Q S}
    {FD.reified.distance Alignment.P Alignment.Q '=:' 1 S}
   end
in
   Root = Sat#Alignment
   {FD. distinct Alignment}
   {FD.sum {Map Preferences Satisfied} '=:' Sat}
   Alignment.franek <: Alignment.beata     % elim. symetr.
   {FD.distribute generic(order:naive value:max) [Sat]}
   {FD.distribute split Alignment}
end
```