Programowanie w Logice

Globalne ograniczenia kombinatoryczne

Przemysław Kobylański

Globalne ograniczenia kombinatoryczne

Co to jest?

- Ograniczenie globalne, to takie, które wiąże wiele zmiennych jednocześnie.
- Można też powiedzieć, że ograniczenie globalne to takie, które wiąże nieustaloną na stałe liczbę zmiennych (lista zmiennych jest często jego argumentem).
- Ograniczenie globalne można wyrazić jako koniunkcję wielu prostszych ograniczeń.
- Zaletą ograniczeń globalnych, w przeciwieństwie do koniunkcji prostszych ograniczeń, jest to, że dostarczają silnikowi przetwarzającemu ograniczenia dużo więcej informacji o strukturze problemu.

Globalne ograniczenia kombinatoryczne

Ograniczenia arytmetyczne

sum(+Vars, +Rel, ?Expr)
Suma zmiennych pozostaje w zadanej relacji (#=, #\=, #<, #>, #>=, #=<) z wartością wyrażenia.

```
?- [A, B, C] ins 0..sup, sum([A, B, C], #=, 100).
A in 0..100,
A+B+C#=100,
B in 0..100,
C in 0..100.
```

Globalne ograniczenia kombinatoryczne

Ograniczenia arytmetyczne

scalar_product(+Cs, +Vs, +Rel, ?Expr)
Iloczyn skalarny wektora współczynników `Cs` z wektorem zmiennych `Vs` pozostaje w zadanej relacji (#=, #\=, #<, #>, #>=, #=<) z wartością wyrażenia.

```
?- [A, B, C] ins 0..sup,
   scalar_product([1, 2, 3], [A, B, C], #=, 100).
A in 0..100,
A+2*B+3*C#=100,
B in 0..50,
C in 0..33.
```

Globalne ograniczenia kombinatoryczne

Ograniczenia arytmetyczne

chain(+Vars, +Relation)
Wartości zmiennych tworzą łańcuch ze względu na relację (#=, #<, #>, #>=, #=<).

```
?- chain([X, Y, Z], #>=).
X#>=Y,
Y#>=Z.
```

Globalne ograniczenia kombinatoryczne

Ograniczenia arytmetyczne

all_different(+Vars)
Zmienne są parami różne.

```
?- [A, B, C] ins 1..2, all_different([A, B, C]).
A in 1..2,
all_different([A, B, C]),
B in 1..2,
C in 1..2.
```

Globalne ograniczenia kombinatoryczne

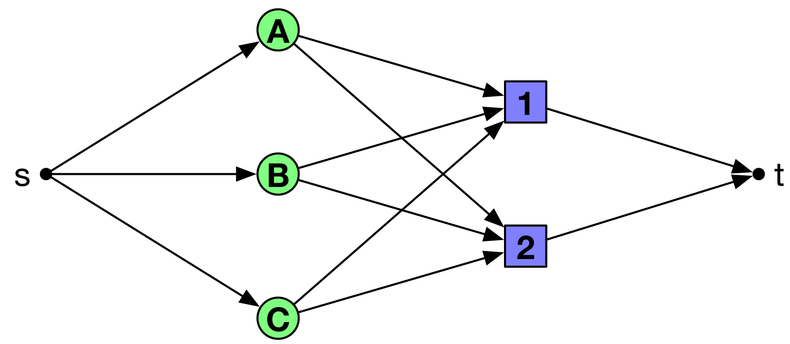
Ograniczenia arytmetyczne

all_distinct(+Vars)
Silniejsza wersja ograniczenia, w którym zmienne są parami różne.

```
?- [A, B, C] ins 1..2, all_distinct([A, B, C]).
false.
```

Globalne ograniczenia kombinatoryczne

Ograniczenia arytmetyczne



Każdy łuk ma przepustowość 1. Czy istnieje przepływ od s do t wielkości 3?

Globalne ograniczenia kombinatoryczne

Ograniczenia arytmetyczne

lex_chain(+Lists)
Listy są w porządku leksykograficznym.

```
?- [A, B, C] ins 1..2,
   lex_chain([[A, B], [B, C], [C, A]]).
A in 1..2,
B#>=A,
lex_chain([[A, B], [B, C], [C, A]]),
freeze(A, clpfd:lex_le([A, B], [B, C])),
B in 1..2,
C#>=B,
freeze(B, clpfd:lex_le([B, C], [C, A])),
C in 1..2.

?- [A, B, C] ins 1..2,
   lex_chain([[A, B], [B, C], [C, A]]), B #= 2.
A = B, B = C, C = 2.
```

Globalne ograniczenia kombinatoryczne

Ograniczenia arytmetyczne

element(?N, +Vars, ?V)
N-ty element listy zmiennych jest równy `V`.
Warunek $element(N, [E_1, E_2, \ldots, E_n], V)$ jest równoważny:

$$N \in 1..n \land \forall_{i \in 1..n}(E_i \neq V \rightarrow N \neq i).$$

```
?- element(N, [E1, E2], V).
N in 1..2,
element(N, [E1, E2], V),
N  #\= 2 #<==> _7386,  N  #\= 1 #<==> _7410,
E1 #\= V #<==> _7434,  E2 #\= V #<==> _7458,
_7458 in 0..1, _7458 #==> _7386, _7386 in 0..1,
_7434 in 0..1, _7434 #==> _7410, _7410 in 0..1.
```

Globalne ograniczenia kombinatoryczne

Ograniczenia arytmetyczne

global_cardinality(+Vars, +Pairs)
Lista par składa się z termów postaci `Key-Num`. Predykat narzuca ograniczenie, że wśród wartości na liście `Vars`, wartość `Key` występuje dokładnie `Num` razy.

```
?- L = [A, B, C, D], % i-ty mówi ile razy występuje i-1
   global_cardinality(L, [0-A, 1-B, 2-C, 3-D]),
   label(L).
L = [1, 2, 1, 0],
A = C, C = 1,
B = 2,
D = 0 ;
L = [2, 0, 2, 0],
A = C, C = 2,
B = D, D = 0 ;
false.
```

Globalne ograniczenia kombinatoryczne

Relacja

tuples_in(+Tuples, +Relation)
Wszystkie listy na liście `Tuples` są elementami listy `Relation`. Każdy element listy `Tuples` jest listą liczb całkowitych lub zmiennych a każdy element listy `Relation` jest listą liczb całkowitych.

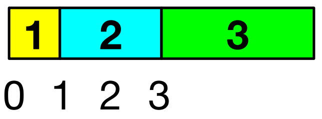
```
?- tuples_in([[X,Y]], [[1,2],[1,5],[4,0],[4,3]]), X = 4.
X = 4,
Y in 0\/3.
```

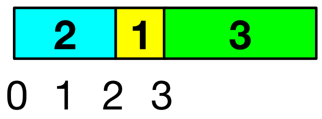
Globalne ograniczenia kombinatoryczne
Układanie harmonogramów

serialized(+Starts, +Durations)
Dane n zadań o chwilach rozpoczęcia $Starts = [S_1, S_2, \ldots, S_n]$ (lista zmiennych lub liczb całkowitych) i czasach trwania $Durations = [D_1, D_2, \ldots, D_n]$ (lista nieujemnych liczb całkowitych) wykonują nie nachodząc się w czasie, tj.

$$\forall_{i \in 1..n-1} \forall_{j \in i+1..n} (S_i + D_i \leq S_j \vee S_j + D_j \leq S_i).$$

```
?- length(Vs, 3), Vs ins 0..3, serialized(Vs, [1,2,3]),
   label(Vs).
Vs = [0, 1, 3] ;
Vs = [2, 0, 3] ;
false.
```





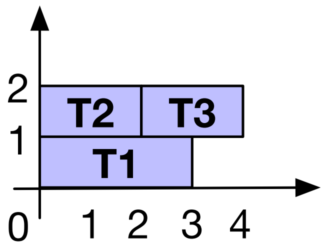
Globalne ograniczenia kombinatoryczne
Układanie harmonogramów

cumulative(+Tasks, +Options)
Lista zadań składa się z elementów postaci $task(S_i, D_i, E_i, R_i, ID_i)$, gdzie S_i jest chwilą rozpoczęcia i-tego zadania, D_i czasem jego trwania, E_i chwilą zakończenia, R_i liczbą jednostek zasobu jaką wymaga do wykonania, ID_i jego identyfikatorem (każde jest zmienną lub nieujemną liczbą całkowitą).
Lista opcji zawiera $limit(L)$, gdzie L jest limitem dostępnych jednostek zasobu (domyślnie $L = 1$).
Ograniczenie zapewnia, że w żadnej chwili łączna liczba jednostek zasobu użytych do wykonania trwających w niej zadań nie przekracza limitu L.

Globalne ograniczenia kombinatoryczne
Układanie harmonogramów

```
tasks_starts(Tasks, [S1,S2,S3]) :-
    Tasks = [task(S1,3,_,1,_), task(S2,2,_,1,_),
             task(S3,2,_,1,_)].

?- tasks_starts(Tasks, Starts), Starts ins 0..10,
   cumulative(Tasks, [limit(2)]), label(Starts).
Tasks = [task(0,3,3,1, _G36), task(0,2,2,1, _G45), ...],
Starts = [0, 0, 2] .
```



Globalne ograniczenia kombinatoryczne
Układanie harmonogramów

Example (Harmonogram zadań)

Czasy trwania zadań i wymagane przez nie jednostki zasobu:

```
tasks([ %D  R
        [2, 1],
        [3, 2],
        [4, 2],
        [3, 3],
        [3, 1],
        [3, 4],
        [5, 2]]).
```

Liczba dostępnych jednostek zasobu:

```
resources(5).
```

Globalne ograniczenia kombinatoryczne

Układanie harmonogramów

Example (Harmonogram zadań cd.)

```
schedule(H, Ss, MS) :-
    tasks(L),
    resources(R),
    MS in 0..H,
    mt(L, H, T, Ss, MS),
    cumulative(T, [limit(R)]),
    once(labeling([min(MS), ff], [MS | Ss])).

mt([], _, [], [], _).
mt([[D, R] | L1], H, [task(S, D, E, R, _) | L2],
                        [S | L3], MakeSpan) :-
    S in 0..H,
    E #= S + D, MS #>= E,
    mt(L1, H, L2, L3, MS).
```

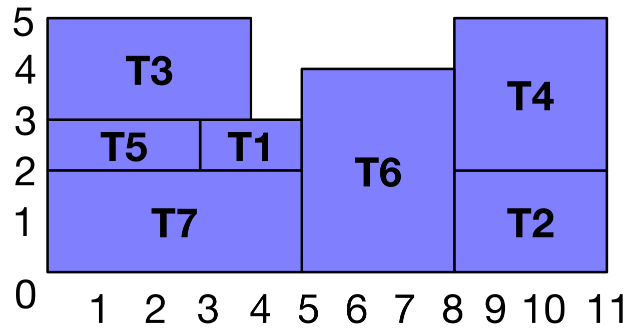
Globalne ograniczenia kombinatoryczne

Układanie harmonogramów

Example (Harmonogram zadań cd.)

Przykładowe zapytanie:

```
?- schedule(20, S, MS).
S = [3, 8, 0, 8, 0, 5, 0],
MS = 11.
```



Globalne ograniczenia kombinatoryczne

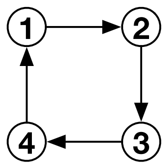
Ograniczenia teoriografowe i geometryczne

circuit(+Vars)
Dana jest lista *n* zmiennych o dziedzinach będących zakresami od 1 do *n*. Jeśli wartością *i*-tej zmiennej jest *k*, to oznacza, że istnieje łuk od węzła *i* do węzła *k*. Ograniczenie gwarantuje, że wszystkie łuki tworzą cykl Hamiltona (cykl przechodzący przez wszystkie *n* węzłów i przez każdy węzeł dokładnie jeden raz). Innymi słowy, permutacja `Vars` ma dokładnie jeden cykl.

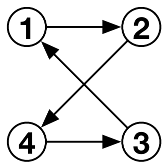
```
?- length(X, 4), circuit(X), label(X).
X = [2, 3, 4, 1] ;
X = [2, 4, 1, 3] ;
X = [3, 1, 4, 2] ;
X = [3, 4, 2, 1] ;
X = [4, 1, 2, 3] ;
X = [4, 3, 1, 2] ;
false.
```

Globalne ograniczenia kombinatoryczne

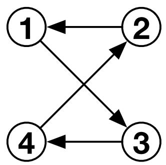
Ograniczenia teoriografowe i geometryczne



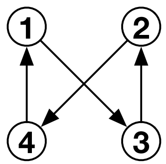
[2, 3, 4, 1]



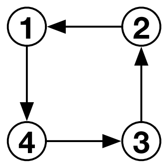
[2, 4, 1, 3]



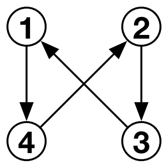
[3, 1, 4, 2]



[3, 4, 2, 1]



[4, 1, 2, 3]



[4, 3, 1, 2]

Globalne ograniczenia kombinatoryczne

Ograniczenia teoriografowe i geometryczne

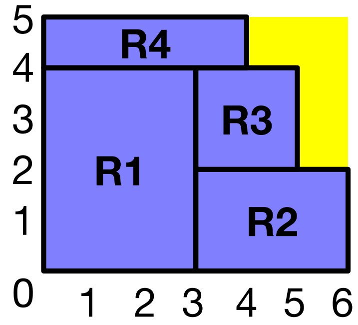
disjoint2(+Rectangles)

Dana lista prostokątów, które reprezentowane są termami postaci $F(X_i, W_i, Y_i, H_i)$, gdzie F jest dowolnym funktorem, (X_i, Y_I) są współrzędnymi lewego dolnego rogu i-tego prostokąta a W_i i H_i są jego, odpowiednio, szerokością i wysokością. Ograniczenie gwarantuje, że wszystkie prostokąty nie nachodzą na siebie (są rozłączne w $2D$).

```
?- X1 in 0..3, X2 in 0..3, X3 in 0..4, X4 in 0..2,
   Y1 in 0..1, Y2 in 0..3, Y3 in 0..3, Y4 in 0..4,
   disjoint2([f(X1,3,Y1,4), f(X2,3,Y2,2),
              f(X3,2,Y3,2), f(X4,4,Y4,1)]),     % w 6x5
   label([X1,X2,X3,X4,Y1,Y2,Y3,Y4]).
X1 = X4, X4 = Y1, Y1 = Y2, Y2 = 0,
X2 = X3, X3 = 3,
Y3 = 2,
Y4 = 4 ;
...
```

Globalne ograniczenia kombinatoryczne

Ograniczenia teoriografowe i geometryczne



Globalne ograniczenia kombinatoryczne

Języki regularne

automaton(+Vars, +Nodes, +Arcs)

Definiuje ograniczenie w postaci automatu skończonego o zadanych węzłach reprezentujących stany i łukach odpowiadających przejściom między stanami[1]

Automaty skończone i rozstrzygane przez nie języki regularne omawiane są na obowiązkowym kursie **Języki Formalne i Techniki Translacji**.

[1] Istnieje też wersja ośmioargumentowa z licznikami.

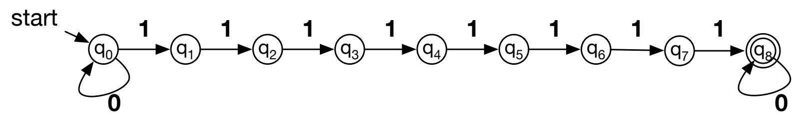
Globalne ograniczenia kombinatoryczne

Języki regularne

Example (Odcinek)

Zdefiniujemy ograniczenie dopuszczające na liście tylko wartości zero-jedynkowe a dodatkowo, jedynki tworzą zwarty odcinek długości 8 (jak w zadaniu 3. z listy 8.).

Wyrażenie regularne opisujące słowa spełniające ten warunek jest postaci $0^*1^80^*$. Odpowiada mu następujący automat skończony:



Example (Odcinek cd.)

Predykat `odcinek(X)` definiuje taki automat:

```
odcinek(X) :-
    automaton(X, [source(q0), sink(q8)],
              [arc(q0, 0, q0),
               arc(q0, 1, q1), arc(q1, 1, q2),
               arc(q2, 1, q3), arc(q3, 1, q4),
               arc(q4, 1, q5), arc(q5, 1, q6),
               arc(q6, 1, q7), arc(q7, 1, q8),
               arc(q8, 0, q8)]).
```


Example (Odcinek cd.)

Przykład użycia:

```
?- length(X, 16), odcinek(X), label(X), writeln(X), fail.
[0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1]
[0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,0]
[0,0,0,0,0,0,1,1,1,1,1,1,1,1,0,0]
[0,0,0,0,0,1,1,1,1,1,1,1,1,0,0,0]
[0,0,0,0,1,1,1,1,1,1,1,1,0,0,0,0]
[0,0,0,1,1,1,1,1,1,1,1,0,0,0,0,0]
[0,0,1,1,1,1,1,1,1,1,0,0,0,0,0,0]
[0,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0]
[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0]
false.
```


labeling(+Options, +Vars)

- Często konieczne jest dobranie odpowiednich strategii sterowania etykietowaniem.
- Opcje predykatu `labeling/2` można podzielić na cztery kategorie:
  - **wybór zmiennej** jaka jest kolejność etykietowanych zmiennych?
  - **porządek wartości** jaka jest kolejność podstawianych? wartości
  - **strategia podziału** w jaki sposób odbywa się podział problemu na podproblemy?
  - **optymalizacja** jakie jest kryterium optymalizacji?
- Z każdej kategorii może wystąpić na liście `Options` co najwyżej jedna opcja.

- Wybór zmiennej:

leftmost	w kolejności na liście (domyślne)
ff	w kolejności rosnącego rozmiaru dziedziny
ffc	jak w przypadku **ff** a gdy dziedziny równoliczne, to najpierw zmienna występująca w większej liczbie ograniczeń
min	w kolejności rosnącego kresu dolnego dziedziny
max	w kolejności malejącego kresu górnego dziedziny

```
?- freeze(X, writeln(x)), freeze(Y, writeln(y)),
   X in 1..20, Y in 5..25, labeling([min], [X, Y]).
x
y
X = 1, Y = 5 .
?- freeze(X, writeln(x)), freeze(Y, writeln(y)),
   X in 1..20, Y in 5..25, labeling([max], [X, Y]).
y
x
X = 1, Y = 5 .
```

Globalne ograniczenia kombinatoryczne
Sterowanie etykietowaniem

- Porządek wartości:

  up — porządek rosnący (domyślne)
  down — porządek malejący

```
?- length(X, 1), X ins 1..3, labeling([up], X).
X = [1] ;
X = [2] ;
X = [3].
?- length(X, 1), X ins 1..3, labeling([down], X).
X = [3] ;
X = [2] ;
X = [1].
```

Globalne ograniczenia kombinatoryczne
Sterowanie etykietowaniem

- Strategia podziału:

  step — dla każdej zmiennej `X` dokonuje się wybór między `X = V` a `X #\= V`, gdzie `V` jest wybraną wartością (domyślne)
  enum — dla każdej zmiennej dokonuje się kolejno wybór $X = V_1, X = V_2, \ldots$ gdzie $V_1, V_2, \ldots$ jest wybranym porządkiem wartości z dziedziny
  bisect — dla każdej zmiennej `X` dokonuje się wybór między `X #=< M` a `X #> M`, gdzie `M` jest wartością środkową w dziedzinie zmiennej `X`

Globalne ograniczenia kombinatoryczne
Sterowanie etykietowaniem

- Optymalizacja:

  min(Expr) — generowanie rozwiązań w kolejności rosnącej wartości `Expr`
  max(Expr) — generowanie rozwiązań w kolejności malejącej wartości `Expr`

Globalne ograniczenia kombinatoryczne
Sterowanie etykietowaniem

Example (Niebijące się hetmany)

Predykat `hetmany(N, P)` narzuca na permutację złożoną z liczb od 1 do N warunek, że N hetmanów ustawionych na szachownicy $N \times N$ nie będzie się biło nawzajem.

```
hetmany(N, P) :-
    length(P, N),    P ins 1..N,
    all_distinct(P), bezpieczna(P).

bezpieczna([]).
bezpieczna([I | L]) :-
    bezpieczna(L, I, 1), bezpieczna(L).

bezpieczna([], _, _).
bezpieczna([J | L], I, K) :-
    abs(I-J) #\= K, K1 is K+1, bezpieczna(L, I, K1).
```

Example (Niebijące się hetmany cd.)

Porównanie czasów etykietowania:

```
?- hetmany(20, X), time(labeling([], X)).
% 155,159,652 inferences, 14.292 CPU in 14.353 seconds (10(
X = [1, 3, 5, 2, 4, 13, 15, 12, 18|...] .

?- hetmany(20, X), time(labeling([ff], X)).
% 318,400 inferences, 0.033 CPU in 0.034 seconds (96% CPU,
X = [1, 3, 5, 14, 17, 4, 16, 7, 12|...] .
```

Uzyskano 433-krotne przyspieszenie czasu znalezienia rozwiązania.