Логическое программирование. Примеры.

Лектор:
доцент каф. АОИ
Салмина Нина Юрьевна

Пример. Размещение 8 ферзей.

Необходимо расставить 8 ферзей на шахматной доске так, чтобы ни один из ферзей не находился под боем другого.

Позиция на доске ферзя: X,Y ∈ [1,2,3,4,5,6,7,8]

Можно зафиксировать исходную позицию:

[1, Y1, 2, Y2, 3, Y3, 4, Y4, 5,Y5, 6, Y6, 7, Y7, 8, Y8]

Пример. Размещение 8 ферзей. Обобщение задачи

1. Список ферзей пуст: решение найдено (нападений в этом случае нет)
2. Список ферзей не пуст:

   [X, Y | L]

   L – список остальных ферзей.

Условия выполнения решения:

1. Ферзи из списка L не должны бить друг друга
2. Ферзь на поле X,Y не должен бить ни одного ферзя из списка L

Пример. Размещение 8 ферзей. Выбор координаты Y

$Y \in [1,2,3,4,5,6,7,8]$

member (Y, [1,2,3,4,5,6,7,8])

```
member ( X, [X | _ ] ).
member ( X, [ _ | T ] ) :- member ( X, T).
```

Пример. Размещение 8 ферзей. Выполнение условия «не бьет»

```
not_beat ( _, _, [ ]).
not_beat ( X,Y, [X1,Y1 | L]) :-
        ? ,
        ? ,
        not_beat (X, Y, L).
```

Пример. Размещение 8 ферзей. Выполнение условия «не бьет»

```
not_beat ( _, _, [ ]).
not_beat ( X,Y, [X1,Y1 | L]) :-
        Y <> Y1,
        abs(Y1-Y) <> abs(X1-X),
        not_beat (X, Y, L).
```

Пример. Размещение 8 ферзей. Решение

```
solution ( [ ] ).
solution( [X, Y | L]) :-
        solution (L),
        member (Y, [1,2,3,4,5,6,7,8]),
        not_beat (X,Y,L).
```

```
?- solution ([1,Y1,2,Y2,3,Y3,4,Y4,5,Y5, 6, Y6, 7, Y7, 8, Y8]).
Y1=4, Y2=2, Y3=7, Y4=3, Y5=6, Y6=8, Y7=5, Y8=1
Y1=5, Y2=2, Y3=4, Y4=7, Y5=3, Y6=8, Y7=6, Y8=1
Y1=3, Y2=5, Y3=2, Y4=8, Y5=6, Y6=4, Y7=7, Y8=1   . . .
92 Solutions
```

Метод «образовать и проверить»

find (X) :- generate (X), test (X).

generate (X) – генерация множества предполагаемых решений задачи

test (X) – проверка предполагаемого решения

Пример. Размещение 8 ферзей. Второй вариант

Формируем только список координат

Y ∈ [1,2,3,4,5,6,7,8]

```
solution1(S) :- another ([1,2,3,4,5,6,7,8], S),
                it_may_be (S).
```

1. another – делаем перестановку списка координат
2. it_may_be – проверяем, допустима ли данная перестановка

Перестановка элементов списка

Включение элемента в список на произвольное место

```
insert (X, T, [X | T]).
insert (X, [Y | YT], [Y | ZT]) :- insert (X, YT, ZT).
```

Построение всех перестановок списка

```
another ([ ], [ ]).
another ( [X | L], P):- another (L, L1),
                        insert ( X, L1, P).
```

Проверка существования решения

Проверка допустимости перестановки

```
it_may_be ([ ]).
it_may_be ([X | L]) :- it_may_be ( L ),
                       not_beat1(X, L, 1).
```

Ферзь не бьет остальные

```
not_beat1 (_, [ ], _).
not_beat1 (Y, [Y1 | L], DX) :- abs(Y-Y1)<>DX,
                               DX1 = DX+1,
                               not_beat1 (Y, L, DX1).
```

Конкурс исполнителей

На Международном конкурсе музыкальных исполнителей зрители заспорили, участник какой страны будет победителем. Были высказаны следующие предположения:

- Первым будет пианист из России, а вторым – пианист из Израиля;
- Исполнитель из России будет вторым, а из Польши – третьим;
- Вторым будет пианист из Германии, а польский музыкант будет четвертым.

Оказалось, в каждом высказывании одно предположение истинно, другое ложно.

Определить победителей.

Конкурс исполнителей.

```
another ([ ],[ ]).
another ([X | L], P) :- another(L, L1), insert (X, L1, P).

may_be ([A, B, C, D]) :- predp1(    ,    ), predp2(    ,    ), predp3(    ,    ).
predp1(X,Y) :-
predp1(X,Y) :-
predp2(X,Y) :-
predp2(X,Y) :-
predp3(X,Y) :-
predp3(X,Y) :-

question(L,S) :- another(L,S), may_be(S).
```

- Первым будет пианист из России, а вторым – пианист из Израиля;
- Исполнитель из России будет вторым, а из Польши – третьим;
- Вторым будет пианист из Германии, а польский музыкант будет четвертым.

goal

```
question ([rus, isr, ger, pol], Y).
```

Конкурс исполнителей.

```
another ([ ],[ ]).
another ([X | L], P) :- another(L, L1), insert (X, L1, P).

may_be ([A, B, C, D]) :- predp1(A, B), predp2(B, C), predp3(B, D).
predp1(X,Y) :- X=rus, not(Y=isr).
predp1(X,Y) :- not(X=rus), Y=isr.
predp2(X,Y) :- X=rus, not(Y=pol).
predp2(X,Y) :- not(X=rus), Y=pol.
predp3(X,Y) :- X=ger, not(Y=pol).
predp3(X,Y) :- not(X=ger), Y=pol.

question(L,S) :- another(L,S), may_be(S).
```

- Первым будет пианист из России, а вторым – пианист из Израиля;
- Исполнитель из России будет вторым, а из Польши – третьим;
- Вторым будет пианист из Германии, а польский музыкант будет четвертым.

goal

```
question ([rus, isr, ger, pol], Y).
```

Результат

Y = ["rus", "ger", "pol", "isr"]

1 Solution

Расположение терминальных ветвей:
подсчет суммы и кол-ва элементов списка
допустимо / ? допустимо

```
sum ( [ ], 0, 0).
sum ([X | T], S, K) :- sum (T, S1, K1),
                       S=S1+X, K=K1+1.
```

```
sum ([X | T], S, K) :- sum (T, S1, K1),
                       S=S1+X, K=K1+1.
sum ( [ ], 0, 0).
```

Расположение терминальных ветвей: подсчет суммы и количества элементов списка

(НЕ)допустимо (?) / допустимо

```
sum ( _, S, S, K, K).
sum ([X | T], SP, S, KP, K) :- !, SP1=SP+X,
                    KP1=KP+1,
                     sum (T, SP1, S, KP1, K).
```

```
sum ([X | T], SP, S, KP, K) :- !, SP1=SP+X,
                    KP1=KP+1,
                     sum (T, SP1, S, KP1, K).
sum ( _, S, S, K, K).
```

Расположение терминальных ветвей: удаление из списка элемента X допустимо / ?

```
delete(X, [X|T], T).
delete(X, [Y|T], [Y|T1]):-delete( X, T, T1).
```

```
delete(X, [Y|T], [Y|T1]):-delete( X, T, T1).
delete(X, [X|T], T).
```

Напечатать список по 5 элементов в строке: допустимо / ?

```
writelist (NL) :- nl, write5 (NL, 0), nl.
```

```
write5 (L, 5) :- !, nl, write5(L, 0).
write5 ( [H | T], N) :- !, write(H, “  “), N1=N+1,
                    write5 (T, N1).
```

```
write5 ( [H | T], N) :- !, write(H, “  “), N1=N+1,
                    write5 (T, N1).
write5 (L, 5) :- !, nl, write5(L, 0).
```

Пример
Фермер, волк, коза и капуста

Фермеру необходимо переправиться через реку. С ним находятся волк, коза и капуста.

Есть лодка, на которой за один раз переправиться могут только двое, причем править лодкой может только фермер.

На одном берегу (без фермера) нельзя оставлять:

волка и козу;

козу и капусту.

Определить способ переправки всех на другой берег без потерь.

Фермер, волк, коза и капуста
Используемые предикаты

```
domains
% место – положение участника на одном из берегов реки
  LOC   =  east;  west
% состояние – расположение участников на берегах реки
  STATE = state (LOC farmer, LOC wolf, LOC goat, LOC cabbage)
% путь – список состояний
  PATH  = STATE*

predicates
  move (STATE, STATE)          % переезд на противоположный берег
  opposite (LOC, LOC)          % изменение места на противоположное
  unsafe (STATE)               % недопустимые состояния
  path (STATE, STATE, PATH, PATH)      % нахождение пути
  write_path (PATH)
  write_move (STATE, STATE)
  go(STATE, STATE)             % целевое правило
```

Фермер, волк, коза и капуста
Перемещение на другой берег

```
% фермер + волк
move (state(X, X, G, C), state (Y, Y, G, C)) :- opposite (X, Y).
% фермер + коза
move (state (X, W, X, C), state (Y, W, Y, C)) :- opposite (X, Y).
% фермер + капуста
move (state (X, W, G, X), state (Y, W, G, Y)) :- opposite (X, Y).
% только фермер
move (state (X, W, G, C), state (Y, W, G, C)) :- opposite (X, Y).

  opposite (east, west).
  opposite (west, east).

  unsafe (state (F,  ,  , _)) :-        % волк ест козу
  unsafe (state (F, _,  ,  )) :-        % коза ест капусту
```

Фермер, волк, коза и капуста
Перемещение на другой берег

```
% фермер + волк
move (state(X, X, G, C), state (Y, Y, G, C)) :- opposite (X, Y).
% фермер + коза
move (state (X, W, X, C), state (Y, W, Y, C)) :- opposite (X, Y).
% фермер + капуста
move (state (X, W, G, X), state (Y, W, G, Y)) :- opposite (X, Y).
% только фермер
move (state (X, W, G, C), state (Y, W, G, C)) :- opposite (X, Y).

  opposite (east, west).
  opposite (west, east).

  unsafe (state (F, X, X, _)) :- opposite (F, X),!.      % волк ест козу
  unsafe (state (F, _, X, X)) :- opposite (F, X),!.      % коза ест капусту
```

Фермер, волк, коза и капуста. Целевое правило

```
% X -  начальное состояние; Y – конечное состояние

go (X,Y):-
    path ( X, Y, [X], Path),
    write ("A solution is:\n"),
    write_path (Path).

goal
go (state (east, east, east, east),
      state (west, west, west, west)).
```

Фермер, волк, коза и капуста
Формирование пути

```
path (Y, Y, Path, Path).      % достигли конечного состояния

% P – формируем промежуточный путь
path (X, Y, P, Path) :-
        move (X, Next_X),          % выбираем перемещение
        not (unsafe (Next_X)),   % проверяем его допустимость
        % проверяем, чтобы такого состояния не было прежде
        not (member (Next_X, P)),
         path (Next_X, Y, [Next_X | P], Path).
```

Фермер, волк, коза и капуста Вывод найденных перемещений на печать

```
write_path ( [H1, H2 | T] ) :-
      write_move (H1, H2), write_path ([H2 | T]), !.
write_path ([ _ ]).

write_move (state (X, W, G, C), state (Y, W, G, C)) :- !,
   write ("The farmer crosses the river from ", X, " to ", Y), nl.
write_move (state (X, X, G, C), state (Y, Y, G, C)):- !,
   write ("The farmer takes the Wolf from ", X, " of the river to ", Y), nl.
write_move (state (X, W, X, C), state (Y, W, Y, C)) :- !,
   write("The farmer takes the Goat from ", X, " of the river to ", Y), nl.
write_move (state (X, W, G, X), state (Y, W, G,Y)) :- !,
   write ("The farmer takes the cabbage from ", X, " of the river to ", Y),
     nl.
```

Решение (2 варианта)

A solution is:
The farmer takes the Goat from west of the river to east
The farmer crosses the river from east to west
The farmer takes the cabbage from west of the river to east
The farmer takes the Goat from east of the river to west
The farmer takes the Wolf from west of the river to east
The farmer crosses the river from east to west
The farmer takes the Goat from west of the river to east

A solution is:
The farmer takes the Goat from west of the river to east
The farmer crosses the river from east to west
The farmer takes the Wolf from west of the river to east
The farmer takes the Goat from east of the river to west
The farmer takes the cabbage from west of the river to east
The farmer crosses the river from east to west
The farmer takes the Goat from west of the river to east

Программирование повторяющихся операций

```
proc_repeat :- «повторяемые операции», fail.
proc_repeat.
```

```
repeat :- person (X, _, _),
        age (data (16, 10, 2020), X, Y),
        write(X,"  ",Y), nl, fail.
repeat.
```