Тема 6. Логические задачи, программирование второго порядка

Логические задачи

1. Задача о переправе через реку.

К берегу реки подошёл человек, который вёл с собой волка, овцу и капусту. У берега реки он обнаружил лодку. В лодку с собой он может взять только кого-то одного. Может ли он перебраться на другой берег со своим скарбом, если волк, оставшись с овцой, съест овцу, а овца, оставшись с капустой, съест капусту?

```
% Ограничения
constraint(wolf, sheep).
constraint(sheep, cabbage).

% Проверка ограничений
test([X, Y]):- not(constraint(X, Y)), not(constraint(Y, X)).

% Запуск программы за N шагов (количество шагов нужно, чтобы
избавиться от зацикливания и обеспечить перебор всех вариантов в
рамках заданного числа шагов)
go(N):-move([wolf, sheep, cabbage], [], [], r, N).

% Вывод результата P. Параметры: (Левый_берег, Правый_берег,
История_движения, Текущее_направление{l,r}, Количество_шагов)
move([], _, P, _, _) :- write(P).

% При первом ходе слева можно оставить тех, кто не съест друг
друга, а налево вернуться пустым
move([X,Y,Z],[], P, r, N):-
N > 0, M is N - 1,
(test([X, Z]), move([X, Z], [Y], [l, r(Y)|P], r, M);
test([Y, Z]), move([Y, Z], [X], [l, r(X)|P], r, M);
test([X, Y]), move([X, Y], [Z], [l, r(Z)|P], r, M)).

% При втором ходе направо можно выбрать любого
move([X,Y],[Z], P, r, N):-
N > 0, M is N - 1,
(move([Y], [X,Z], [r(X)|P], l, M);
move([X], [Y,Z], [r(Y)|P], l, M)).

% При возврате налево пустым можно возвращаться только если на
правом берегу друг друга не съедят, иначе можно выбрать любого
move([X],[Y,Z], P, l, N):-
N > 0, M is N - 1,
(test([Y,Z]), move([X],[Y,Z], [l|P], r, M);
move([Y, X], [Z], [l(Y)|P], r, M);
```

```
move([Z, X], [Y], [l(Z)|P], r, M)).

% При движении направо получаем последний ход
move([X],[Y,Z], P, r, N):-
N > 0, M is N - 1,
move([], [X, Y, Z], [r(X)|P], _, M).
```

Решения:

```
?- go(7).
[r(sheep),l,r(cabbage),l(sheep),r(wolf),l,r(sheep)]
[r(sheep),l,r(wolf),l(sheep),r(cabbage),l,r(sheep)]
```

2. Задача об островитянах – рыцарях и лжецах.

Перед нами три островитянина A, B и C, о каждом из которых известно, что он либо рыцарь (всегда говорит правду), либо лжец. Двое из них (A и B) высказывают следующие утверждения:

A: Мы все лжецы.

B: Один из нас рыцарь.

Кто из трех островитян A, B и C рыцарь и кто лжец?

```
% Утверждения островитян
told(a, [lier, lier, lier]).
told(b, [knight, lier, lier]).
told(b, [lier, knight, lier]).
told(b, [lier, lier,  knight]).

% Либо A-рыцарь и его утв. верно, либо он лжец и лжёт
check(a, [A, B, C]) :-
A= knight, told(a, [A, B, C]);
A= lier, (B= knight; C= knight; B= knight, C= knight).

% Либо B-рыцарь и его утв. верно, либо он лжец и лжёт
check(b, [A, B, C]) :-
B= knight, told(b, [A, B, C]);
B= lier, (A= lier, C= lier; A= knight, C= knight).

% Необходимо проверить утверждения A и B
go([A, B, C]) :- check(a, [A, B, C]), check(b, [A, B, C]).
```

Решение:

```
[lier,knight,lier]
```

Другой вариант решения этой же задачи (про островитян):

```
% Перечисляем утверждения островитян A и B
firstOpinion(lier, lier, lier).
```

```
secondOpinion(knight, lier, lier).
secondOpinion(lier, knight, lier).
secondOpinion(lier, lier, knight).

% Перебираем варианты лжец/рыцарь для каждого из островитян
heIs(lier).
heIs(knight).

% Ищем решение
go(A, B, C) :-
heIs(A), heIs(B), heIs(C), testOpinions(A, B, C).

% Проверяем утверждения первого и второго
% Если он лжец, то его утверждение ложно
% Если он рыцарь, то его утверждение истинно
testOpinions(A, B, C) :-
(A == lier, not(firstOpinion(A, B, C)) ;
A == knight, firstOpinion(A, B, C)),
(B == lier, not(secondOpinion(A, B, C)) ;
B == knight, secondOpinion(A, B, C)).
```

Решение:

```
A = lier
B = knight
C = lier
```

3. Задача о сыновьях.

Четверо друзей – Алексей Иванович, Федор Семенович, Валентин Петрович и Григорий Аркадьевич пошли с детьми в парк культуры и отдыха. В парке они решили прокатиться на «колесе обозрения». В кабинах колеса оказались вместе: Алексей Иванович с Леней, Андрей с отцом Коли, Дима с отцом Андрея, Федор Семенович с сыном Валентина Петровича, Валентин Петрович с сыном Алексея Ивановича. Кто является чьим сыном?

```
fathers([ai, fs, vp, ga]).
sons([l, k, a, d]).

check(L):-member(trio(ai, _, l), L),
member(trio(FatherK, k, _), L), member(trio(FatherK, _, a), L),
member(trio(FatherA, a, _), L), member(trio(FatherA, _, d), L),
member(trio(vp, SonVP, _), L), member(trio(fs, _, SonVP), L),
member(trio(ai, SonAI, _), L), member(trio(vp, _, SonAI), L).

generate([trio(F1, S1, C1), trio(F2, S2, C2), trio(F3, S3, C3),
trio(F4, S4, C4)]):-
fathers([F1, F2, F3, F4]),
sons(S), permutation(S, [S1, S2, S3, S4]),
```

```
sons(C), permutation(C, [C1, C2, C3, C4]).

go(L):-generate(L), check(L).
```

Решение:

```
L = [trio(ai,d,l),trio(fs,k,a),trio(vp,a,d),trio(ga,l,k)]
```

4. Задача о бегуне-лгуне

Четверо ребят - Алексей, Борис, Иван и Григорий – соревновались в беге. На следующий день на вопрос, кто какое место занял, они ответили так:

Алексей : «Я не был ни первым, ни последним».

Борис : «Я не был последним».

Иван : «Я был первым».

Григорий : «Я был последним».

Известно, что три из этих ответов правильные, а один-неверный.

Требуется написать программу, которая определила бы, кто сказал неправду и кто был первым на самом деле.

```
alex([_,a,_,_]).
alex([_,_,a,_]).
boris([b,_,_,_]).
boris([_,b,_,_]).
boris([_,_,b,_]).
ivan([i,_,_,_]).
grig([_,_,_,g]).

go(L):-permutation([a,b,i,g], L),
(alex(L), boris(L), ivan(L), not(grig(L));
alex(L), boris(L), not(ivan(L)), grig(L);
alex(L), not(boris(L)), ivan(L), grig(L);
not(alex(L)), boris(L), ivan(L), grig(L)).
```

Решения:

L = [b,a,i,g];

L = [b,i,a,g]

<u>Программирование второго порядка</u>

Пролог в классической постановке находит только одно решение, один раз доказывает цель. Если же мы хотим получить все возможные доказательства, то в интерпретаторе мы нажимаем клавишу «;». Существуют методы получения всех результатов доказательства цели. Эти методы относятся к **программированию второго порядка**. (Первый порядок – одно решение, второй порядок – все решения в виде списка).

В качестве предикатов второго порядка в Прологе используются следующие: `bagof(X, G, L)` и `setof(X, G, L)`. Такие предикаты называются **предикатами**

второго порядка или множественными предикаты.

Предикат `bagof(X, G, L)` доказывает цель G, в которую в качестве переменной входит единственная не унифицированная переменная X, столько раз, сколько это возможно, в результате в список L помещаются все значения, которые принимал X в результате доказательства цели G. Пример:

```
age(peter, 7).
age(ann, 5).
age(tom, 12).
age(alex, 5).
?- bagof(X, age(X,5), L).
L = [ann, alex]
yes
```

Если переменная X будет не единственная, то будут анализироваться комбинации неунифицированных переменных, например:

```
?- bagof(X, age(_, X), L).
L = [5] ? ;
L = [5] ? ;
L = [7] ? ;
L = [12]
yes
```

Предикат `setof` делает то же самое, что и `bagof` за одним исключением, в списке L элементы будут упорядочены по возрастанию и все повторы, если таковые есть, исключены. Для приведённой выше программы вопрос будет выглядеть следующим образом:

```
?- setof(X, age(X, 5), L).
L = [alex, ann]
yes
```

Предлагается написать аналогичный предикат `findall` с теми же параметрами, решающий те же задачи, что и `bagof` за одним исключением: он должен собирать все возможные решения в независимости от количества неунифицированных переменных в G. Решение:

```
findall(X, G, _) :-
     assertz(result([])),    % Создаёт пустой список результата
     call(G),                % (Пере)доказывает цель
     retract(result(L)),     % Достаёт накопленный список
     assertz(result([X|L]))  % Добавляет в список новый X
,                            % Предлагает передоказать цель
     fail.
findall(_, _, List) :-       % Получает результаты
     retract(result(List)).
```

Встроенные предикаты для определения типов переменных:

- `var(X)` – X – не унифицированная переменная;
- `nonvar(X)` – X – не переменная;
- `atom(X)` – X – атом;
- `integer(X)` – X – целое число;
- `real(X)` – X – вещественное число;

- `atomic(X)` – X – атом или число;
- `number(X)` – X – число.

Как можно описать предикат `number`?

Ответ: `number(X) :- Y is X + 0.`

Если X не является числом, то Пролог ответит `no`.

Программа для сложения `plus(X, Y, Z)` эквивалентная X + Y = Z:

```
plus(X, Y, Z) :- nonvar(X), nonvar(Y), Z is X + Y.
plus(X, Y, Z) :- nonvar(X), nonvar(Z), Y is Z - X.
plus(X, Y, Z) :- nonvar(Z), nonvar(Y), X is Z - Y.
```

В чём отличие данной программы от классической для Пролога записи Z is X + Y?