

Министерство образования Республики Беларусь  
Учреждение образования  
“Белорусский государственный университет  
информатики и радиоэлектроники”  
Факультет информационных технологий и управления  
Кафедра интеллектуальных информационных технологий

**Лабораторная работа N3**  
по дисциплине «Логические основы интеллектуальных систем»

Выполнил  
студент группы  
721702

Клюев А.А.

Проверил

Бобков А.В.

Минск 2020

**Тема:** Программирование на языке Prolog решение одной из представленных задач.

**Цель:** В соответствии с вариантом необходимо реализовать программу решающую поставленную задачу. В результате выполнения программы должен выводиться протокол работы, описывающий решение задачи. Необходимо предусмотреть возможность простого задания начального и целевого состояний задачи, а также параметризацию числовых характеристик этих состояний. При описании состояний решаемой задачи использовать термы или списки. Рекомендуется использовать Visual Prolog или SWI Prolog. Достаточно реализовать консольное приложение.

**Вариант задания** — 4. Обезьяна сидит в клетке, в центре под потолком висит банан, который можно достать, взяв палку и встав на подставленный ящик. Ящик и палка находятся в углу клетки, обезьяна – в другом углу. Обезьяна может двигать ящик, брать палку и сбивать палкой в руке банан при достаточной высоте. Матрицей задаётся положение обезьяны, бананов, палки и ящика. Можно добавить непроходимые участки в клетке, например жуткие лужи. Требуется для произвольного исходного состояния найти последовательность операций, при которой обезьяна достаёт банан.

#### Ход выполнения:

Опишем основные факты и правила написанной программы:

1. Перечислим все возможные корректные элементы матриц(списков) и правило для проверки корректности элемента матрицы(списка) и правило для проверки всех элементов списка:

```
spec_el([]).
spec_el(nl).

elem([my]).
elem([sk]).
elem([bx]).
elem([ba]).
elem([my, sk]).
elem([my, bx]).
elem([my, ba]).
elem([sk, bx]).
elem([sk, ba]).
elem([bx, ba]).
elem([my, sk, bx]).
elem([my, sk, ba]).
elem([my, bx, ba]).
elem([sk, bx, ba]).
elem([my, sk, bx, ba]).

valid_el(E) :-
    ( elem(E)
    ; spec_el(E)
    ).

check_elem([]).
check_elem([E|T]) :-
    valid_el(E),
    check_elem(T).
```

2. Для решения задачи, нужно, чтобы в одной ячейке одновременно находились обезьяна, палка, коробка и банан:

```
solution(S) :-
```

```
member([my, sk, bx, ba], S).
```

3. Определим правило для проверки корректности матрицы, представляющей клетку с обезьяной. Правильной является такая матрица, в которой все ячейки корректные, в конце находится символ переноса строки, в матрице нет повторяющихся элементов на уровне подписков, а также в ней присутствуют все главные акторы: обезьяна, палка, коробка и банан.

```
valid(A) :-  
    check_elem(A),  
    length(A, L),  
    I is L - 1,  
    nth01(I, A, nl),  
    flatten(A, FA),  
    delete(FA, nl, DFA),  
    is_set(DFA),  
    member(my, FA),  
    member(sk, FA),  
    member(bx, FA),  
    member(ba, FA).
```

4. Опишем правила перемещения обезьяны по клетке:
- Перемещение описывается двумя матрицами одинакового размера. Обе матрицы правильные, в соответствии с правилами выше.
  - Были сформированы следующие основные правила:
    - Когда обезьяна перемещает предметы, она их должна переносить в одной клетке с собой
    - Ни одно перемещение не может происходить без обезьяны
    - Банан никогда не перемещается, даже с обезьяной
    - Обезьяна не может одновременно двигать коробку и палку
    - Обезьяна может подобрать палку или коробку, только переместившись на клетку с предметом
    - Если конечное состояние — решение задачи, то перед этим должно быть состояние, когда либо у обезьяны — палка, а под бананом — коробка, либо наоборот.
  - Правила перемещения обезьяны по координатам:
    - У матриц должны быть на одинаковых позициях символы переноса строки
    - Обезьяна не может перемещаться в ячейку с символом переноса строки
    - В зависимости от того, находится ли справа или слева от ячейки с обезьяной в начальной позиции символ переноса строки, вычисляется список возможных координат перемещения обезьяны. Если координата ячейки с обезьяной в конечной позиции входит в сформированный список, то правило принимается.
  - Если все правила выше будут удовлетворены, правило move примет значение true.

```
move(A, B) :-  
    length(A, L),  
    length(B, L),  
    valid(A),  
    valid(B),  
    % Interaction rules  
    ( % Not throwing items rule  
        ( member(C, A), C = [my|_], nth01(I, A, C),  
          nth01(I, B, D), D \= [my|_],  
          member(M, D), \+ member(M, C)
```

```

-> false
; true
),
% Not moving without monkey rule
( member(C, A), C \= [my|_], nth01(I, A, C),
  nth01(I, B, D), D \= [my|_],
  C \= D
-> false
; true
),
% Not moving banana rule
( member(C, A), member(ba, C),
  nth01(I, A, C), nth01(I, B, D),
  \+ member(ba, D)
-> false
; true
),
% Moving box and stick separately rule
( SB = [[my, sk, bx], [my, sk, bx, ba]],
  member(C, SB), member(D, SB),
  nth01(I1, A, C), nth01(I2, B, D), I1 \= I2
-> false
; true
),
% Grabbing stick rule
( nth01(I1, A, [sk|_]), nth01(I2, B, [my, sk|_])
-> I1 = I2
; true
),
% Grabbing box rule
( nth01(I1, A, [bx|_]), nth01(I2, B, [my, bx|_])
-> I1 = I2
; true
),
% Grabbing banana rule
( solution(B)
-> ( subset([[my, sk], [bx, ba]], A)
    ; subset([[my, bx], [sk, ba]], A)
  )
; true
)
),
% Moving rules
( nth01(TSZ, A, n1), nth01(TSZ, B, n1),
  M1 = [my|_], M2 = [my|_],
  nth01(IA, A, M1), nth01(IB, B, M2),
  not(nth01(IB, A, n1)), not(nth01(IA, B, n1)),
  S1 is IA + 1, SM1 is IA - 1,
  S3 is IA + TSZ, SM3 is IA - TSZ,
  S4 is S3 + 1, SM4 is SM3 - 1,
  S5 is S4 + 1, SM5 is SM4 - 1,
  ( nexttto(n1, M1, A)
-> ( nexttto(M1, n1, A)
    -> false

```

```

; memberchk(IB, [S1, S4, SM4, S5, SM5])
)
; ( nextto(M1, nl, A)
-> memberchk(IB, [SM1, S4, SM4, S5, SM5])
; memberchk(IB, [S1, SM1, S3, SM3, S4, SM4, S5, SM5])
)
)
).

```

5. Опишем основные правила формирования списка перемещений для решения задачи.

Правило `check_path` описывает возможность перехода в следующее состояние и отсутствие такого перехода в пути решения (необходимо для устранения заикливаний в решении).

Правило `find_way` формирует список, элементами которого будут матрица полей решения задачи. Изначально в этом списке только начальное состояние. Для каждого состояния, в соответствии с правилом проверки перехода формируются все возможные переходы и для первого по списку идёт нахождение решения дальше. Если не удалось найти решение по данному переходу, программа вернётся и начнёт поиск решения для следующего состояния. Таким образом получается дерево переходов с прямым обходом в глубину, до того момента, как не будет найдено состояние, при котором задача решена. Если в начале списка оказывается решение задачи, то список переворачивается, на экран выводится сообщение о том, что задача решена и выводится сам ход решения.

```

check_path(A, B, Way) :-
    move(A, B),
    not(member(B, [A|Way])).

find_way([[Result|Way]|_], Result) :-
    reverse([Result|Way], RWay),
    write('Solved!'),
    nl,
    print_way(RWay).
find_way([[Temp|Way]|B], Result) :-
    ( setof([Next, Temp|Way],
            check_path(Temp, Next, Way),
            A),
      append(B, A, Way1),
      !,
      find_way(Way1, Result)
    ; find_way(B, Result)
    ).

```

6. Для запуска решателя используется следующее правило. В соответствии с ним, будет выведено только одно решение задачи

```

run(Start, Result) :-
    find_way([[Start]], Result),
    !.

```

## Тестирование программы:










### 1. Ввод:

```

run([
    [my], [ba], [sk, bx], nl
], [
    [], [my, sk, bx, ba], [], nl
]).












































```

### Выход:

## 2. Вход:

## Выход:

																																																																																																																																									
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--

## Вывод:

В ходе данной лабораторной работы была разработана программа, решающая задачу про “обезьяну в клетке” для матриц полей разных разных размерностей. Для реализации задачи использовался SWI Prolog. Программа была протестирована на работоспособность и были получены верные решения поставленной задачи.