Функциональное и логическое программирование

Лекция 7

2.12 Решение логических задач с использованием списков

2.12.1 Задача о фермере, волке, козе и капусте

Фермер (**farmer**), волк (**wolf**) , коза (**goat**) и капуста (**cabbidge**) находятся на одном берегу. Всем надо перебраться на другой берег на лодке. Лодка перевозит только двоих. Нельзя оставлять на одном берегу козу и капусту (коза съест капусту), козу и волка (волк съест козу).

Процесс перевозки - последовательность состояний state с четырьмя аргументами, каждый из которых может принимать 2 значения: left или right и отражает соответственно нахождение фермера, волка, козы и капусты.

Например, state(left,right,left,right) отражает, что фермер и коза находятся на левом берегу, а волк и капуста – на правом.

Фермер может перевести на другой берег, кроме себя, либо волка, либо козу, либо капусту.

Для описания перевозки определим предикат move, описывающий переходы из одного состояния в другое. Для переправления с одного берега на другой введем предикат opposite, который определяет сторону берега, противоположную исходной.

Например, перевозка волка на другой берег:

```
move(state(X,X,G,C),state(Y,Y,G,C)):-opposite(X,Y).
```

С помощью предиката danger определим опасные состояния (волк и коза на одном берегу, а фермер на другом или коза и капуста на одном берегу, а фермер на другом).

Предикат path формирует список из последовательности состояний, решающих поставленную задачу.

Будем добавлять в результирующий список новое состояние, если в него возможен переход из текущего состояния, оно не опасное и не содержится в сформированной части списка (для избегания зацикливания).

```
goal:-S=state(l,l,l,l),G=state(r,r,r,r),
      path(S,G,[S],L),reverse(L,L1),writeln(L1),fail.
goal.
move(state(X,X,G,C),state(Y,Y,G,C)):-opposite(X,Y).
move(state(X,W,X,C),state(Y,W,Y,C)):-opposite(X,Y).
move(state(X,W,G,X),state(Y,W,G,Y)):-opposite(X,Y).
move(state(X,W,G,C),state(Y,W,G,C)):-opposite(X,Y).
opposite(l,r).
opposite(r,l).
danger(state(X,Y,Y,_)):-opposite(X,Y).
danger(state(X,_,Y,Y)):-opposite(X,Y).
path(G,G,L,L).
path(S,G,L,L1):-move(S,S1), not(danger(S1)),not(member(S1,L)),
                path(S1,G,[S1|L],L1).
```

```
?- goal.
[state(l,l,l,l),state(r,l,r,l),state(l,l,r,l),state(r,r,r,l),state(l,r,l,l),state(r,r,l,r),state(l,r,l,r),state(r,r,r,r)]
[state(l,l,l,l),state(r,l,r,l),state(l,l,r,l),state(r,l,r,r),state(l,l,l,r),state(r,r,l,r),state(l,r,l,r),state(r,r,r,r)]
true.
```

.

2.12.2 Решение числовых ребусов

$$\begin{array}{r} \text{APAPA} \\ +\ \text{SLEEPY} \\ \hline \text{ETUDES} \end{array}$$

Задача состоит в том, чтобы заменить все буквы цифрами. Одинаковым буквам должны соответствовать одинаковые цифры, а разным буквам – разные цифры.

Определим предикат `sum(N1,N2,N)`, который истинен, если существует такая замена букв цифрами в словах `N1`, `N2`, `N`, что `N1+N2=N`.

Числа будем представлять списком цифр, из которых оно состоит. Будем считать, что все списки имеют одинаковую длину.

При выполнении сложения столбиком суммирование цифр чисел выполняется справа налево с учетом переноса из предыдущего разряда. Следовательно, нужно хранить информацию о переносе из предыдущего разряда и о переносе в следующий разряд.

Разным буквам должны соответствовать разные цифры, значит при суммировании цифр нужна информация о цифрах, доступных до и после сложения (эта информация будет представляться двумя списками). Заведем вспомогательный предикат sum:

sum(N1,N2,N, P_before,P_after, Cifri_before,Cifri_after)

Определим вспомогательный предикат sum:

- Если у всех трех чисел имеется хотя бы одна цифра, то суммируем числа без самой левой цифры и, используя список оставшихся цифр и перенос из предыдущего разряда, суммируем самые левые цифры и добавляем полученную сумму в результат суммирования без самой левой цифры. Для суммирования цифр напишем предикат sumc.

- Если все три списка пустые, то переносов разрядов нет, и списки цифр до и после сложения совпадают. Это соответствует правилу остановки рекурсии.

Для суммирования цифр определим предикат sumc.

sumc(D1,D2,D,B, A,LB,LA)

D1,D2 – цифры, которые складываются

D – цифра результата в этом разряде

B – перенос из предыдущего разряда

A – перенос в следующий разряд

LB – список цифр для выбора до сложения

LA - список цифр для выбора после сложения

Первые три аргумента предиката должны быть цифрами. Если какая-нибудь из этих переменных не конкретизирована, то ее необходимо конкретизировать какой-нибудь цифрой из списка LB. После конкретизации цифру следует удалить из списка доступных цифр. Если переменная уже имела значение, то удалять из списка доступных цифр ничего не надо.

Для получения значения неозначенной переменной будем использовать предикат choice.

Если цифра не конкретизирована, то ее надо конкретизировать цифрой из списка цифр, которые еще не использовались, и удалить эту цифру из списка доступных цифр. Если цифра конкретизирована (проверка предикатом nonvar), то список доступных цифр не изменятся.

```
sum(N1,N2,N):-sum(N1,N2,N,0,0,[0,1,2,3,4,5,6,7,8,9],_).
sum([],[],[],0,0,L,L).
sum([D1|N1],[D2|N2],[D|N],C1,C,L1,L):-
     sum(N1,N2,N,C1,C2,L1,L2),
     sumc(D1,D2,D,C2,C,L2,L).
sumc(D1,D2,D,C2,C,L2,L):-choise(D1,L2,L3),
                         choise(D2,L3,L4),
                         choise(D,L4,L),
                         S is D1+D2+C2,
                         D is S mod 10,
                         C is S//10.
choise(X,L,L):-nonvar(X),!.
choise(X,[X|T],T).
choise(X,[Y|T],[Y|T1]):-choise(X,T,T1).
```

?- sum([0,A,P,A,P,A], [S,L,E,E,P,Y], [E,T,U,D,E,S]).

```
?- sum([0,A,P,A,P,A],[S,L,E,E,P,Y],[E,T,U,D,E,S]).
A = 2,
P = 8,
S = 5,
L = 7,
E = 6,
Y = 3,
T = 0,
U = 4,
D = 9 ;
false.
```

```
  БАЙТ
+ БАЙТ
-------
 СЛОВО
```

```
?- sum([0,Б,А,Й,Т],[0,Б,А,Й,Т],[С,Л,О,В,О]).
Б = 3,
А = 6,
Й = 4,
Т = 1,
С = 0,
Л = 7,
О = 2,
В = 8
Unknown action: ж (h for help)
Action?
Unknown action: ж (h for help)
Action? ;
Б = 4,
А = 8,
Й = 1,
Т = 3,
С = 0,
Л = 9,
О = 6,
В = 2 ;
Б = 7,
А = 8,
Й = 2,
Т = 3,
С = 1,
Л = 5,
О = 6,
В = 4 ;
Б = 3,
А = 9,
Й = 1,
Т = 4,
С = 0,
Л = 7,
О = 8,
В = 2
.....
```

Этот ребус имеет 14 решений.

2.13 Строки

Строка – последовательность символов, заключенная в двойные кавычки.

Предикаты для работы со строками:

1. string_length(S, L)

Определяет длину строки S.

Пример 1:

string_length("abc d", N).

N = 5.

2. string_concat(S1,S2,S3)

Соединяет две строки S1 и S2 в третью S3. Можно использовать для разбиения строки.

Пример 2:

string_concat("ab", "12", S).

S = "ab12"

string_concat(S1, S2, "abc").

S1 = "", S2 = "abc";

S1 = "a", S2 = "bc";

...

3. sub_string(S,K,N,R,S1)

Выделяет в строке S подстроку S1, которая начинается с K-го элемента и содержит N символов. R – количество символов, стоящих в S после подстроки S1. Нумерация элементов строки начинается с 0.

Пример 3:

sub_string("abcdefg", 3, 4, R, S).

S = "defg", R = 0

sub_string("abcdefg", -, -, 3, S).

S = "abcd";

S = "bcd";

S = "cd";

S = "d";

S = "";

false

4. string_chars(S,L)

Преобразует строку S в список символов и наоборот.

Пример 4:

```
string_chars(S, ['A', '1', 'a']).
  S = "A1a".
```

5. string_to_list(S,L)

Преобразует строку S в список кодов символов и наоборот.

Пример 5:

```
?- string_to_list("asdf",L).
L = [97, 115, 100, 102].

?- string_to_list(S,[97, 115, 100, 102]).
S = "asdf".
```

6. char_code(C,K)

Преобразует символ C в его код K и наоборот.

Пример 6:

```
char_code('a',K).
K = 97.
```

7. split_string(S,R,D,L)

Преобразует строку S в список подстрок L, используя R как разделитель, удаляя из начала и конца подстрок символы строки D.

Пример 7:

```
?- split_string("as,    gh: f!,  gh."," ",",:.!",L).
L = ["as", "", "", "", "gh", "f", "", "gh"].

?- split_string("as,    gh: f!,  gh."," ",",:.! ",L).
L = ["as", "gh", "f", "gh"].
```

8. atomic_list_concat(L,R,S)

Преобразует список L в строку S, используя R как разделитель и наоборот.

Пример 8:

```
?- atomic_list_concat(["asdd",12,'asd',df]," ",S).
S = 'asdd 12 asd df'.
```

9. number_string(N,S)

Преобразует число N в строку S и наоборот.

Пример 9:

```
number_string(3,S).
    S = "3".
```

10. atom_string(A,S)

Преобразует атом A в строку S и наоборот.

Пример 10:

```
atom_string(a,S).
 S = "a".
```

Пример 11:

Предикат, который преобразует строку S в строку S1, удаляя все пробелы.

```
del(S,S1):-string_chars(S,L),delete(L,' ',L1), string_chars(S1,L1).
```

```
?-  del("as    gh ,    h",S).
S = "asgh,h".
```

Пример 12:

Предикат, который считает количество вхождений символа C в строку S.

```
goal1:-writeln('Строка? '),read(S),writeln('Символ? '),read(C),
      string_chars(S,L),счет(L,C,N),writeln(N).
счет([],_,0).
счет([C|T],C,N):-счет(T,C,N1),N is N1+1.
счет([_|T],C,N):-счет(T,C,N).
```

```
?- goal1.
Строка?
|: "asdf aaaghj a".
Символ?
|: 'a'.
5
true .
```

2.14 Предикаты для работы с файлами

1. exists_file(<‘имя файла’>)

Завершается успешно, если файл с указанным именем существует (обратные слэши дублируются).

2. open(<'имя файла’>, <режим>,F)

Открытие файла для чтения, записи или добавления.

Режимы:

- read (для чтения);
- write (для записи);
- append (для добавления).

F – файловая переменная

Предикаты для работы с файлами являются внелогическими. Чтение и обработку данных следует выполнять отдельно!

3. set_input(F)

   set_output(F)

Перенаправление ввода из файла или вывода в файл.

4. close(F)

Закрытие файла.

5. see(<'имя файла‘>)

   tell(<'имя файла‘>)

Открытие и перенаправления ввода из файла или вывода в файл вместо 2 и 3.

При перенаправлении ввода на клавиатуру, а вывода на экран в качестве имени файла используют имя user.

6. seen

   told

Закрытие файлов, открытых с помощью see и tell.

7. seeing(F)

   telling(F)

Связывает F с именем файла, являющегося текущим входным или выходным потоком.

8. at_end_of_stream

Успешно завершается, если найден конец файла.

9. read_line_to_codes(F,L)

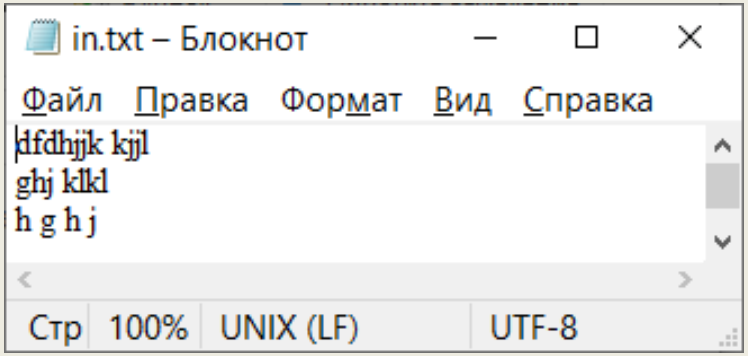
Читает строку из входного потока F и преобразует ее в список кодов символов этой строки (без кода перевода строки).

10. read_stream_to_codes(F,L)

Читает содержимое из входного потока F (до конца файла) и преобразует его в список кодов символов (включая коды перевода строки 10).

Пример 1: Вывести содержимое текстового файла на экран.

```
goal2:-see('in.txt'), seeing(F),
      read_stream_to_codes(F,L),
      string_to_list(S,L),
      write(S),
      seen.
```




```
?- goal2.
dfdhjjk kjjl
ghj klkl
h g h j
true.
```