



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 10

По дисциплине «Функциональное и логическое программирование»

Студент: Тимонин А. С.

Группа ИУ7-626

Преподаватель Толпинская Н. Б.

Москва.
2020 г.

Практическая часть

Ответить на вопросы (коротко):

1. Как организуется хвостовая рекурсия в Prolog?
2. Какое первое состояние резольвенты?
3. Каким способом можно разделить список на части, какие, требования к частям?
4. Как выделить за один шаг первые два подряд идущих элемента списка?
Как выделить 1-й и 3-й элемент за один шаг?
5. Как формируется новое состояние резольвенты?
6. Когда останавливается работа системы? Как это определяется на формальном уровне?

Используя хвостовую рекурсию, разработать, комментируя аргументы, эффективную программу, позволяющую:

1. Сформировать список из элементов числового списка, больших заданного значения;
2. Сформировать список из элементов, стоящих на нечетных позициях исходного списка (нумерация от 0);
3. Удалить заданный элемент из списка (один или все вхождения);
4. Преобразовать список в множество (можно использовать ранее разработанные процедуры).

Убедиться в правильности результатов

Для одного из вариантов ВОПРОСА и 1-ого задания составить таблицу, отражающую конкретный порядок работы системы:

Т.к. резольвента хранится в виде стека, то состояние резольвенты требуется отображать в столбик: вершина – сверху! Новый шаг надо начинать с нового состояния резольвенты! Для каждого запуска алгоритма унификации, требуется указать № выбранного правила и соответствующий вывод: успех или нет –и почему.

Листинг 1. Реализация программы

```
domains
    i = integer
    mlist = i*.

predicates
    nondeterm newListAboveNum(mlist, integer, mlist).
    nondeterm isAbove(integer, integer).

    nondeterm newListEvenPos(integer, mlist, mlist).
    nondeterm oddp(integer).

    nondeterm deleteNumInList(mlist, integer, mlist).

    nondeterm removeRepeatedElements(integer, mlist, mlist).
    nondeterm makeSet(mlist, mlist).

clauses
    %append([], _).
    %append([Head|Tail],[Head|NewTail]) :- isAbove(Head, 3), append(Tail, NewTail),
    !.
    %append([_|Tail],[_|NewTail]) :- append(Tail, NewTail), !.

    isAbove(CompNum, Num) :- CompNum > Num, !.

    newListAboveNum([], _, Result):-Result = [],!.
    newListAboveNum([Head|Tail], Num, [Head|NewTail]) :- isAbove(Head, Num),
                                                         newListAboveNum(Tail, Num,
                                                         NewTail), !.

    newListAboveNum([_|Tail], Num, NewTail) :- newListAboveNum(Tail, Num,
                                                         NewTail), !.

    oddp(A) :-
        B = A mod 2, B = 1, !;
        B = A mod 2, B = -1, !.

    newListEvenPos(_, [], Result):-Result = [],!.
    newListEvenPos(Index, [Head|Tail], [Head|NewTail]) :- oddp(Index),
                                                         NewIndex = Index+1,
                                                         newListEvenPos(NewIndex, Tail,
                                                         NewTail),
                                                         !.

    newListEvenPos(Index, [_|Tail], NewTail) :- NewIndex = Index+1,
                                                         newListEvenPos(NewIndex, Tail, NewTail),
                                                         !.

    deleteNumInList([], _, Result):-Result = [],!.
    deleteNumInList([Head|Tail], Num, [Head|NewTail]) :- Head <> Num,
                                                         deleteNumInList(Tail, Num,
                                                         NewTail), !.

    deleteNumInList([_|Tail], Num, NewTail) :- deleteNumInList(Tail, Num,
                                                         NewTail), !.

    removeRepeatedElements(_, [], []).
    removeRepeatedElements(ElemSet, [ElemSet | Tail], NewSet) :-
        removeRepeatedElements(ElemSet, Tail, NewSet).

    removeRepeatedElements(ElemSet, [CheckElem | Tail],
        [CheckElem | NewTail]) :-
        ElemSet<>CheckElem,
        removeRepeatedElements(ElemSet, Tail,
NewTail).

    makeSet([], Result):- Result = [], !.
    makeSet([Head|Tail],[Head| NewTail]) :- removeRepeatedElements(Head, Tail,
Set).
```

```

goal
makeSet(Set, NewTail).

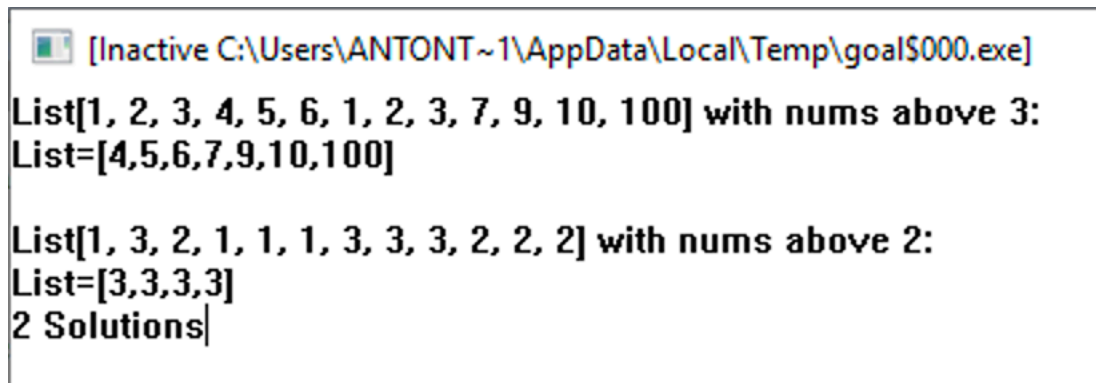
write("List[1, 2, 3, 4, 5, 6, 1, 2, 3, 7, 9, 10, 100] with nums above 3: \n"),
newListAboveNum([1, 2, 3, 4, 5, 6, 1, 2, 3, 7, 9, 10, 100], 3, List);
write("\nList[1, 3, 2, 1, 1, 1, 3, 3, 3, 2, 2, 2] with nums above 2: \n"),
newListAboveNum([1, 3, 2, 1, 1, 1, 3, 3, 3, 2, 2, 2], 2, List).

write("List[[1, 2, 3, 4, 5, 6, 7, 8] with oddp Index: \n"),
newListEvenPos(0, [1, 2, 3, 4, 5, 6, 7, 8], List);
write("\nList[0, 1, 0, 1] with oddp Index: \n"),
newListEvenPos(0, [0, 1, 0, 1], List).

write("Delete 3 in List[3, 3, 3] \n"),
deleteNumInList([3, 3, 3], 3, List);
write("\nDelete 0 in List[1, 0, 2, 0, 0, 3, 4] \n"),
deleteNumInList([1, 0, 2, 0, 0, 3, 4], 0, List).

write("Make set of List[5, 5, 5] \n"),
makeSet([5, 5, 5], Result);
write("\nMake set of List[3, 1, 1, 2, 1, 2] \n"),
makeSet([3, 1, 1, 2, 1, 2], Result).

```



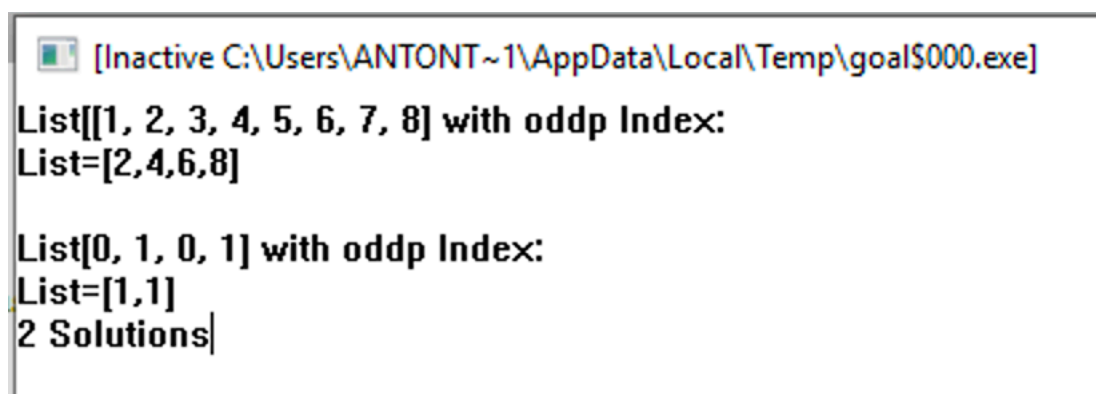
```

[Inactive C:\Users\ANTONT~1\AppData\Local\Temp\goal$000.exe]
List[1, 2, 3, 4, 5, 6, 1, 2, 3, 7, 9, 10, 100] with nums above 3:
List=[4,5,6,7,9,10,100]

List[1, 3, 2, 1, 1, 1, 3, 3, 3, 2, 2, 2] with nums above 2:
List=[3,3,3,3]
2 Solutions

```

Рисунок 1. Тестирование newListAboveNum(*m*list, *integer*, *m*list)



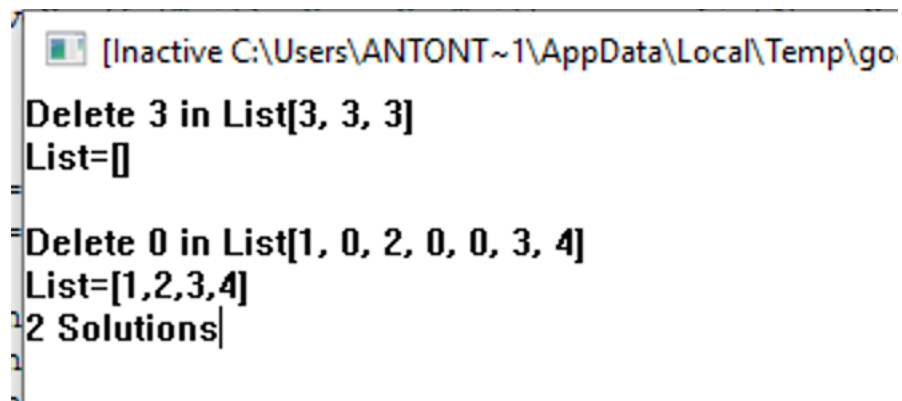
```

[Inactive C:\Users\ANTONT~1\AppData\Local\Temp\goal$000.exe]
List[[1, 2, 3, 4, 5, 6, 7, 8] with oddp Index:
List=[2,4,6,8]

List[0, 1, 0, 1] with oddp Index:
List=[1,1]
2 Solutions

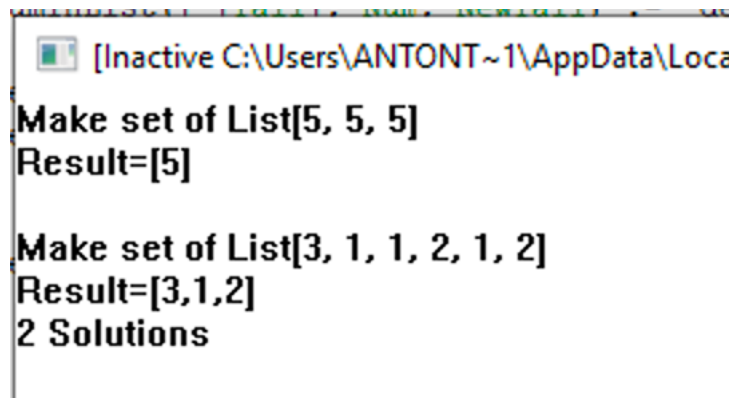
```

Рисунок 2. Тестирование newListEvenPos(*integer*, *m*list, *m*list)



```
[Inactive C:\Users\ANTONT~1\AppData\Local\Temp\go  
Delete 3 in List[3, 3, 3]  
List=[]  
Delete 0 in List[1, 0, 2, 0, 0, 3, 4]  
List=[1,2,3,4]  
2 Solutions
```

Рисунок 3. Тестирование deleteNumInList([mlist](#), [integer](#), [mlist](#))



```
[Inactive C:\Users\ANTONT~1\AppData\Local  
Make set of List[5, 5, 5]  
Result=[5]  
Make set of List[3, 1, 1, 2, 1, 2]  
Result=[3,1,2]  
2 Solutions
```

Рисунок 4. Тестирование makeSet([mlist](#), [mlist](#))

Формирование ответа

Таблица 1.

№ шага	Состояние резольвенты, и вывод: дальнейшие действия (почему?)	Для каких термов запускается алгоритм унификации: $T_1=T_2$ и каков результат	Дальнейшее действие: прямой ход или откат
1			Прямой ход
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			

Теоретическая часть

Как организуется хвостовая рекурсия в Prolog?

Хвостовая рекурсия в Пролог организуется при помощи правила, которое обращается к тому же правилу.

Какое первое состояние резольвенты?

Простой вопрос

Каким способом можно разделить список на части, какие, требования к частям?

Prolog существует более общий способ доступа к элементам списка. Для этого используется метод разбиения списка на начало и остаток. Начало списка – это группа первых элементов, не менее одного. Остаток списка – обязательно список (может быть пустой). Для разделения списка на начало, и остаток используется вертикальная черта (|) за последним элементом начала.

Как выделить за один шаг первые два подряд идущих элемента списка?

```
lis([A | [B | C]]) :- write(A),  
                     write(B),  
                     write(C),  
                     !.
```

```
lis([1, 2, 3, 4, 5]).
```

```
A = 1  
B = 2  
C = [3, 4, 5]
```

Как выделить 1-й и 3-й элемент за один шаг?

```
lis([A | [_ | [B | C]]) :- write(A),  
                          write(B),
```

```
write(C),  
!.  
  
lis([1, 1000, 3, 4, 5, 6, 7, 8, 9]).  
  
A = 1  
B = 3  
C = [4,5,6,7,8,9]
```

Как формируется новое состояние резольвенты?

При возврате отменяется последняя уже выполненная редукция (восстанавливается предыдущее состояние резольвенты) и система выполняет ре- конкретизацию переменных, которые были конкретизированы на предыдущем шаге.

Когда останавливается работа системы? Как это определяется на формальном уровне?

Работа интерпретатора завершается либо когда список инструкций опустеет, либо когда произойдет какая-либо ошибка во время выполнения инструкции.