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

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Лабораторная работа № 20

Формирование и модификация списков на Prolog

Дисциплина	Функциональное и логическое программирование
Студент	Сиденко А.Г.
Группа	ИУ7-63Б
Преподаватель	Толпинская Н.Б., Строганов Ю.В.

Москва, 2020 г.

Задание

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

Программа

```
1  domains
2    list = integer*
3    num = integer
4  predicates
5    over_number(list, num, list).
6    odd_list(list, list).
7    delete_number(list, num, list).
8    member(num, list).
9    multi_list(list, list).
10 clauses
11   over_number([], _, []):-!.
12   over_number([X|Tail], Num, [X|List]):-
13     X > Num,
14     over_number(Tail, Num, List),!.
15   over_number([_|Tail], Num, List):-
16     over_number(Tail, Num, List).
17
18   odd_list([], []):-!.
19   odd_list([_], []):-!.
20   odd_list([_, Second|Tail], [Second|OddTail]):-
21     odd_list(Tail, OddTail).
22
23   delete_number([], _, []):-!.
24   delete_number([Num|Tail], Num, List):-
```

```
25      delete_number(Tail, Num, List),!.
26    delete_number([X|Tail], Num, [X|List]):-
27      delete_number(Tail, Num, List).
28
29    member(Num, [Num|_]):-!.
30    member(Num, [_|Tail]):-
31      member(Num, Tail).
32
33    multi_list([], []):-!.
34    multi_list([Num|Tail], List):-
35      member(Num, Tail),
36      multi_list(Tail, List),!.
37    multi_list([Num|Tail], [Num|List]):-
38      multi_list(Tail, List).
```

Приведем таблицу для задания 1.

```
1  goal
2    over_number([5,1,2,3], 2, List).
```

№ шага	Состояние резольвенты	Сравниваемые термы; результат; подстановка, если есть	Дальнейшие действия: прямой ход или откат
1	*over_number*([5,1,2,3], 2, List)	По *over_number*([5,1,2,3], 2, List)) ищется системой определение отношения (по имени предиката и списку (числу) аргументов)	Определение отношения найдено, заносится в стек *over_number*([5,1,2,3], 2, List), прямой ход
2	5 > 2, *over_number*([1,2,3], 2, List)	Начинает «раскрываться» правило, т.е. доказывается каждое целевое утверждение в теле правила последовательно слева направо X > Num, *over_number*(Tail, Num, List)	Прямой ход
3	*over_number*([1,2,3], 2, List)	5>2	Успешная унификация, переход к следующему целевому утверждению

4	*over_number*([1,2,3], 2, List)	По *over_number*([1,2,3], 2, List)) ищется системой определение отношения (по имени предиката и списку (числу) аргументов)	Определение отношения найдено, заносится в стек *over_number*([1,2,3], 2, List), прямой ход
5	1 > 2, *over_number*([2,3], 2, List)	Начинает «раскрываться» правило, т.е. доказывается каждое целевое утверждение в теле правила последовательно слева направо X > Num, *over_number*(Tail, Num, List)	Прямой ход
6	*over_number*([2,3], 2, List)	1>2	Неуспешная унификация, переход к следующему определению с таким именем
7	*over_number*([2,3], 2, List)	Начинает «раскрываться» правило, т.е. доказывается каждое целевое утверждение в теле правила последовательно слева направо *over_number*(Tail, Num, List)	Прямой ход
8	*over_number*([2,3], 2, List)	По *over_number*([2,3], 2, List)) ищется системой определение отношения (по имени предиката и списку (числу) аргументов)	Определение отношения найдено, заносится в стек *over_number*([2,3], 2, List), прямой ход
9	2 > 2, *over_number*([3], 2, List)	Начинает «раскрываться» правило, т.е. доказывается каждое целевое утверждение в теле правила последовательно слева направо X > Num, *over_number*(Tail, Num, List)	Прямой ход

10	*over_number*([3], 2, List)	2>2	Неуспешная унификация, переход к следующему определению с таким именем
11	*over_number*([3], 2, List)	Начинает «раскрываться» правило, т.е. доказывается каждое целевое утверждение в теле правила последовательно слева направо *over_number*(Tail, Num, List)	Прямой ход
12	*over_number*([3], 2, List)	По *over_number*([3], 2, List)) ищется системой определение отношения (по имени предиката и списку (числу) аргументов)	Определение отношения найдено, заносится в стек *over_number*([3], 2, List), прямой ход
13	$3 > 2$, *over_number*([], 2, List)	Начинает «раскрываться» правило, т.е. доказывается каждое целевое утверждение в теле правила последовательно слева направо X > Num, *over_number*(Tail, Num, List)	Прямой ход
14	*over_number*([], 2, List)	2>2	Успешная унификация, переход к следующему целевому утверждению
15	*over_number*([], 2, List)	По *over_number*([], 2, List)) ищется системой определение отношения (по имени предиката и списку (числу) аргументов)	Определение отношения найдено, заносится в стек *over_number*([], 2, List), прямой ход
16		Унификация *over_number*([], 2, List) с *over_number*([], _, [])	Унификация успешна, List=[], отсечение, возврат из стека *over_number*([], 2, List)
17	Резольента пуста	Поочередно достаем из стека и подставляем X, List	List =[3], достаем из стека *over_number*([3], 2, List), откат

18	Резольента пуста	Поочередно достаем из стека и подставляем X, List	List =[3], достаем из стека $over\_number$([2,3], 2, List), откат
19	Резольента пуста	Поочередно достаем из стека и подставляем X, List	List =[3], достаем из стека $over\_number$([1,2,3], 2, List), откат
20	Резольента пуста	Поочередно достаем из стека и подставляем X, List	List =[5,3], достаем из стека $over\_number$([5,1,2,3], 2, List), откат
21	Резольента пуста		Стек пуст. Вывод результата. Завершение программы.

Вывод

Эффективный способ организации рекурсии – хвостовая рекурсия. Эффективность рекурсивной процедуры повышается благодаря отсечению неперспективных путей поиска решения. Используя «!» – отсечение. Которое сократит количество выполняемых унификаций для достижения максимальной эффективности работы системы.

Ответы на вопросы

1. Как организуется хвостовая рекурсия в Prolog?

   Хвостовая рекурсия: Для ее осуществления рекурсивный вызов определяемого предиката должен быть последней подцелью в теле рекурсивного правила и к моменту рекурсивного вызова не должно остаться точек возврата (непроверенных альтернатив).

2. Какое первое состояние резольвенты?

   Вопрос.

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

   В Prolog существует более общий способ доступа к элементам списка. Для этого используется метод разбиения списка на начало и остаток. Для этого используется вертикальная черта (|) за последним элементом начала.

   Начало списка – это группа первых элементов, не менее одного. Остаток списка – обязательно список (может быть пустой), всегда один.

4. Как выделить за один шаг первые два подряд идущих элемента списка? Как выделить 1-й и 3-й элемент за один шаг?

   Два подряд идущих:

```
1 [First, Second|Tail]
```

   1-й и 3-й:

```
1 [First, _, Third|Tail]
```

5. Как формируется новое состояние резольвенты?

   Резольвента - текущая цель, существующая на любой стадии вычислений. Резольвенты порождаются целью и каким-либо правилом или фактом, которые просматриваются последовательно сверху вниз. Если резольвента существует при наиболее общей унификации, она вычисляется. Если пустая резольвента с помощью такой стратегии не найдена, то ответ на вопрос отрицателен.

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

   Когда стек пуст.