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 № 20

Формирование и модификация списков на Prolog

Студент Лучина Е.Д

Группа ИУ7-61Б

Преподаватель Толпинская Н.Б.

Москва.

апрель 2020 г.

Содержание отчета

Полный текст задания!!!,

Ответы на вопросы,

Текст программы с комментариями обозначений и идентификаторов аргументов,

Таблицу для 1-ого задания, демонстрирующие работу системы при одном из успешных вариантов вопроса.

Выводы: за счет чего достигнута эффективность реализации каждого задания.

Задание

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.

Текст процедуры, Вопрос:….

Шаг	Текущая резольвента (ТР)	ТЦ, выбираемые правила: сравниваемые термы, подстановка	Дальнейшие действия с комментариями
№!	...	...	...

Ответы на вопросы

1. Как организуется хвостовая рекурсия в Prolog?

Хвостовая рекурсия организуется следующим образом: рекурсивный вызов должен быть последней подцелью; нужно избавиться от точек возврата с помощью предиката отсечения (!), чтобы исключить возможные альтернативы.

2. Какое первое состояние резольвенты?

Вопрос записанный в goal

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

Для разделения списка на начало, и остаток используется вертикальная черта (|) за последним элементом начала. Остаток это всегда один (простой или составной) терм. Если начало состоит из одного элемента, то получим: голову и хвост.

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

[X, Y | Z] - X - первый элемент, Y - второй элемент списка, Z - третий. Не интересующие элементы можно обозначить анонимной переменной [X, _, Y | Z]. X - первый элемент, Y - третий, Z - хвост.

5. Как формируется новое состояние резольвенты?

Преобразования резольвенты выполняются с помощью редукции - замены цели телом того правила из базы знаний, заголовок которого унифицируется с целью. (факт - правило с пустым телом). В текущей резольвенте выбирается одна из подцелей (по стековому принципу - верхняя) и для неё выполняется редукция. Затем, к полученной конъюнкции целей применяется полученная подстановка.

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

Когда все знания в БЗ отмечены и резольвента пуста.

Листинг программы

Листинг 1.

Сформировать список из элементов числового списка, больших заданного значения

```
domains
    ilist = integer*

predicates
    filter(ilist, integer, ilist)

clauses
    filter([], _, []) :- !.
    filter([X|Z], A, Res) :-
        X <= A, filter(Z, A, Res), !.
    filter([X|Z], A, Res) :-
        filter(Z, A, Rz),
        Res = [X | Rz].
```

Примеры работы. Тестирование.

`filter([], 6, L).`	`L = []` `1 Solution`
`filter([4, 2, 6, 1, 4], 3, L).`	`L = [4, 6, 4]` `1 Solution`
`filter([1, 4, 2], 6, L).`	`L = []` `1 Solution`
`filter([8, 9, 10], 4, L).`	`L = [8, 9, 10]` `1 Solution`
`filter([4, 5, 6], 5, [6]).`	`yes`
`filter([4, 5, 6], 8, [4, 9]).`	`no`

Первый аргумент - исходный список, второй - заданное значение, третий - результирующий список. Если исходный список пустой, то в независимости от значения второго аргумента, результатом будет пустой список. Если список не пустой, его делим на голову X и хвост Z, рассматриваем первый элемент (голову, X). Если X не больше заданного A, пропустим этот элемент, иначе добавим его в список. Чтобы пропустить элемент вызывается рекурсивная подцель, аргументы которой - хвост исходного списка, то же число A, тот же результат Res. Чтобы добавить элемент, обработаем хвост и результат свяжем с Rz, потом добавим элемент X в начало Res = [X|Rz]. Первый терм (случай пустого списка) является тупиком рекурсии. Для повышения эффективности используем предикат отсечения.

Листинг 2.

Сформировать список из элементов, стоящих на нечетных позициях исходного списка.

```
domains
     ilist = integer*

predicates
     filter(ilist, ilist)

clauses
     filter([],[]) :- !.
     filter([_|[]], []) :- !.
     filter([_, X| Z], Res) :-
          filter(Z, Rz), Res = [X|Rz].
```

Примеры работы. Тестирование.

`filter([], []).`	`yes`
`filter([], [4,5]).`	`no`
`filter([], L).`	`L=[]` `1 Solution`
`filter([4], []).`	`yes`
`filter([4], [4]).`	`no`
`filter([4], L).`	`L=[]` `1 Solution`
`filter([1, 2], L).`	`L=[2]` `1 Solution`
`filter([1, 2, 6], L).`	`L=[2]` `1 Solution`
`filter([4, 3, 2, 1, 5, 7], L).`	`L=[3, 1, 7]` `1 Solution`

Предикат filter в данном случае имеет два аргумента - исходный список и результирующий. Если исходный список пустой, результирующий тоже будет пустой. С остальными термами попытка унификации будет неудачной, поэтому используем предикат отсечения. Если список имеет один элемент, он имеет индекс 0 - четный, результат пустой список. Оба эти предиката являются тупиками рекурсивной процедуры, первый если список имеет четное число элементов, второй - если нечетное. Если список имеет более одного элемента, рекурсивно рассматриваем первые два элемента. Первый элемент нас не

интересует, обозначим его анонимной переменной. В результирующий список добавляем второй элемент так как он имеет нечетный индекс. Обрабатываем хвост, добавляем элемент в начало.

Листинг 3. - Удалить заданный элемент из списка (один или все вхождения)

```
domains
    ilist = integer*

predicates
    filter(ilist, integer, ilist)

clauses
    filter([],_,[]) :- !.
    filter([A|Z], A, Res) :-
        filter(Z, A, Res), !.
    filter([X|Z], A, Res):-
        filter(Z, A, RZ),
        Res = [X|RZ].
```

Примеры работы. Тестирование.

filter([], 3, []).	yes
filter([1, 2, 1], 1, [2]).	yes
filter([], 44, [9, 3, 2]).	no
filter([3, 4, 2], 4, []).	no
filter([], 5, Res).	Res=[] 1 Solution
filter([1, 2, 3], 1, Res).	Res=[2,3] 1 Solution
filter([1, 2, 3], 2, Res).	Res=[1,3] 1 Solution
filter([1, 2, 3], 3, Res).	Res=[1,2] 1 Solution
filter([1,1,1,1], 1, Res).	Res=[] 1 Solution
filter([2, 3, 4], 9, Res).	Res=[2, 3, 4] 1 Solution
filter([1,2,4,2,4,1,3], 2, Res).	Res=[1,4,4,1,3] 1 Solution

Удаление всех вхождений элемента из списка - составление нового списка, пропуская элементы равные заданному значению. Пустой список дает пустой список. Непустой список делим на голову и хвост. Если голова равна A, запускаем рекурсию для хвоста, результирующий список будет таким же как и для хвоста (пропускаем значение A). В конце используем предикат отсечения. Тогда для терма с исходным списком, первый элемент которого - A, не будет запущен процесс унификации с последним правилом. Третье правило будет унифицировано с вопросом, если исходный список вопроса имеет не пустой и первый элемент не равен A. Обработаем хвост списка (рекурсивно) и добавим первый элемент в начало.

Листинг 4. - Преобразовать список в множество

```
domains
     ilist = integer*

predicates
     member(integer, ilist)
     filter(ilist, ilist)
clauses
     member(X, [X|_]):-!.
     member(X, [_|T]):-member(X, T).

     filter([], []):-!.
     filter([X], [X]):-!.
     filter([X|Y], Res):-
          member(X, Y),
          filter(Y, Res), !.
     filter([X|Y], Res):-
          filter(Y, Ry),
          Res = [X|Ry].
```

Примеры работы. Тестирование.

`filter([], []).`	`yes`
`filter([5], []).`	`no`
`filter([], M).`	`M=[]` `1 Solution`
`filter([7], M)`	`M=[7]` `1 Solution`
`filter([5, 5, 5], M).`	`M=[5]` `1 Solution`
`filter([1, 2, 1, 2, 3, 1], M).`	`M=[2,3,1]` `1 Solution`

`filter([1, 2, 1, 2, 3, 1], [1, 2, 3]).`	`no`

Данная программа составляет из списка множество, элементы которого упорядочены в соответствии со своим последним вхождением в список. Результат обработки пустого списка - пустой список. Используем предикат отсечения, для отсечения бесперспективных ветвей поиска решений (остальные термы заведомо не будут успешно унифицированы). Если X входит в хвост Y, пропустим этот элемент. Иначе (это гарантирует предикат отсечения во втором правиле) обработаем хвост и добавим элемент в начало списка.

Поиск принадлежности элемента списку осуществляется с помощью рекурсивной процедуры member. Если искомый элемент - первый, то завершаем поиск. Иначе рассматриваем хвост.

Таблица

Таблица для 1-ого задания, демонстрирующая работу системы при одном из успешных вариантов вопроса.

Текст процедуры (отфильтровать все элементы большие переданного значения) и вопрос.

```
clauses
        filter([], _, []) :- !.
        filter([X|Z], A, Res) :-
                X <= A, filter(Z, A, Res), !.
        filter([X|Z], A, Res) :-
                filter(Z, A, Rz),
                Res = [X | Rz].
goal
   filter([2, 5, 6], 4, L).
```

Таблица 1.

Шаг	Текущая резольвента (ТР)	ТЦ, выбираемые правила: сравниваемые термы, подстановка	Дальнейшие действия с комментариями
№1	filter([2, 5, 6], 4, L).	ТЦ = filter([2, 5, 6], 4, L). унификация с начала базы filter([2, 5, 6], 4, L). != filter([], _, [])	унификация невозможна возврат к ТЦ, метка переносится ниже
		ТЦ = filter([2, 5, 6], 4, L). поиск зания от метки filter([2, 5, 6], 4, L). = filter([X|Z], A, Res) Успешная унификация Подстановка = {X=2, Z=[5, 6], A = 4, L = Res}	Прямой ход. Проверка тела правила. Изменение резольвенты - замена цели на тело правила. Применение подстановки к резольвенте.
№2	2 <= 4, filter([5, 6], 4, L), !.	2 <= 4 - истинно	Прямой ход. Истинная цель удалена из резольвенты.

№3	filter([5, 6], 4, L), !.	ТЦ =filter([5, 6], 4, L). унификация с начала базы filter([5, 6], 4, L) != filter([], _, [])	унификация невозможна возврат к ТЦ, метка переносится ниже
		ТЦ = filter([5, 6], 4, L). поиск зания от метки filter([5, 6], 4, L). = filter([X|Z], A, Res) Успешная унификация Подстановка = {X=5, Z=[6], A = 4, Res = L}	Прямой ход. Проверка тела правила. Изменение резольвенты - замена цели на тело правила. Применение подстановки к резольвенте.
№4	5 <= 4, filter([6], 4, L), !. !.	5 <= 4 ложно	откат
№5	filter([5, 6], 4, L), !.	ТЦ = filter([5, 6], 4, L). поиск зания от метки filter([5, 6], 4, L). = filter([X|Z], A, Res) Успешная унификация Подстановка = {X=5, Z=[6], A = 4, Res = L}	Прямой ход. Проверка тела правила. Изменение резольвенты - замена цели на тело правила. Применение подстановки к резольвенте.
№6	filter([6], 4, Rz), L = [5| Rz]. !.	ТЦ =filter([6], 4, Rz). унификация с начала базы filter([6], 4, L) != filter([], _, [])	унификация невозможна возврат к ТЦ, метка переносится ниже
		ТЦ = filter([6], 4, Rz). поиск зания от метки filter([6], 4, Rz). = filter([X|Z], A, Res) Успешная унификация Подстановка = {X=6, Z=[], A = 4, Res = Rz}	Прямой ход. Проверка тела правила. Изменение резольвенты - замена цели на тело правила. Применение подстановки к резольвенте.
№7	6 <= 4, filter([], 4, Rz), !. L = [5| Rz]. !.	6 <= 4 ложно	откат
№8	filter([6], 4, Rz), L = [5| Rz]. !.	ТЦ = filter([6], 4, Rz). поиск зания от метки filter([6], 4, Rz). = filter([X|Z], A, Res) Успешная унификация Подстановка = {X=6, Z=[], A = 4, Res = Rz}	Прямой ход. Проверка тела правила. Изменение резольвенты - замена цели на тело правила. Применение подстановки к резольвенте.
№9	filter([], 5, Rz2), Rz1= [6| Rz2]. L = [5| Rz1]. !.	ТЦ = filter([], 5, Rz2), Успешная унификация filter([], 5, Rz2) = filter([], _, []) {[]=[], _ = 5, [] = Rz2}	Прямой ход. Редукция резольвенты, применение подстановки.

№10	!. Rz1= [6| []]. L = [5| Rz1]. !.	! - придикат отсечения. Для цели filter([], 5, Rz2) система больше не будет искать решения.	Редукция резольвенты.
№11	Rz1= [6| []]. L = [5| Rz1]. !.	Rz1 = [6|[]] Rz1 = []	Редукция резольвенты, применение подстановки.
№12	L = [5| [6]]. !.	L = [5|[6]] = [5, 6] Для filter([5, 6], 4, L) найдены все возможные решения, отмечены все знаний БЗ.	Редукция резольвенты.
№13	!.	Предикат отсечения означает что для вопроса filter([2, 5, 6], 4, L). больше решений искать не нужно	Редукция резольвенты.
№14	пустая	Завершение работы программы	

Выводы

В Prolog рассматривая вариант работы программы с поиском всех возможных решений, алгоритм поиска решений осуществляет полный перебор всех знаний. Для достижения большей эффективности стоит использовать предикаты отсечения ! и fail. Они позволяют не рассматривать бесперспективные ветви поиска и завершают поиск для данного примера вопроса либо успехом (!) либо неудачей (fail). Также на эффективность влияет формулировка правил и фактов. Если тело правило содержит проверки его применимости, их стоит внести в заголовок в качестве аргументов. Например list(L) :- L = []. Заменить на list([]). Или значение переменной (несущей в себя смысл результата применения правила) не вычисляется, а просто “присваивается” ей, то имеет смысл указать это прямо в заголовке и в момент унификации переменная будет связана с нужным значением. Все это ускоряет процесс унификации, уменьшает количество вычислений и сравнений, избавляет от ненужных переменных, уменьшает количество работы по изменению резольвенты.