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»

Дисциплина: «Функциональное и логическое программирование»

Лабораторная работа №20

Студент: Левушкин И. К.
Группа: ИУ7-62Б
Преподаватели: Толпинская Н. Б.,
Строганов Ю. В.

Москва, 2020 г.

Цель работы

Изучить способы формирования и модификации списков в Prolog, эффективные методы обработки списков и порядок реализации рекурсивных программ.

Задачи работы

Приобрести навыки формирования и модификации списков на Prolog, эффективного способа их обработки, организации и прядка работы соответствующих программ.

Изучить особенность использования переменных при обработке списков. Способ формирования и изменения резольвенты в этом случае и порядок формирования ответа.

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:...

№ шага	Текущая резольвента - ТР	ТЦ, выбираемые правила: сравниваемые термы, подстановка	Дальнейшие действия с комментариями
шаг1	...	...	...
...	...	...	...

Реализация программы: сформировать список из элементов числового списка, больших заданного значения.

```
domains

        lstI = integer*
        number = integer
predicates

        list_create(lstI, number, lstI)
        list_create_help(lstI, number, lstI, lstI)
clauses

        list_create_help([], _, Help, Help).

        list_create_help([H|T], Number, Help, Answer) :-
        H > Number, !, list_create_help(T, Number, [H|Help], Answer).

        list_create_help([_|T], Number, Help, Answer) :-
        list_create_help(T, Number, Help, Answer).

        list_create(List, Number, Answer) :-
        list_create_help(List, Number, [], Answer).
```

Порядок и особенности выполнения программы и формирования результата.

В разделе domains необходимо объявить тип списка (integer*).

Чтобы организовать хвостовую рекурсию, необходимо ввести дополнительный параметр Help, который будет накапливать в себе результат (в данном примере - список из элементов, больших заданного значения), а при выходе из рекурсии выполнить подстановку: Answer = Help.

Выход из рекурсии будет осуществляться с помошью правила list_create_help([], _, Help, Help), когда список будет пуст.

Рекурсию организуют следующие два правила: list_create_help([H|T], Number, Help, Answer) и list_create_help([_|T], Number, Help, Answer).

Первое из них выполняется в том случае, если очередной элемент списка больше заданного значения (H > Number), при этом второе правило не будет выполняться из-за предиката отсечения !.

И, соответственно, второе правило выполнится, когда первое не будет вы-

полняться - H !> Number. Таким образом, элементы списка, которые больше заданного значения, и элементы, которые не больше заданного значения, обрабатываются по-разному (разными правилами).

Первое вставляет текущий элемент в голову результирующего списка, а второе - игнорирует его.

Стоит отметить, что поскольку элементы последовательно добавляются в голову результирующего списка, полученный результат будет перевернутый. Однако, в данном случае нас это не интересует, поскольку в задании заранее не обговаривался порядок, в котором должен был быть сформирован список.

Тесты

Пример 1:

```
goal
        list_create([1,2,3,4,5,6,7,8,9], 5, Answer).
%Вывод:
        Answer=[9,8,7,6]
        1 Solution
```

Пример 2:

```
goal
        list_create([1,2,3,4,5,6,7,8,9], 9, Answer).
%Вывод:
        Answer=[]
        1 Solution
```

Пример 3:

```
goal
        list_create([], 0, Answer).
%Вывод:
        Answer=[]
        1 Solution
```

Порядок работы системы:

№ шага	Текущая резольвента - TP	TЦ, выбираемые правила: сравниваемые термы, подстановка	Дальнейшие действия с комментариями

1	list_create([], 0, Answer).	ТЦ = list_create([], 0, Answer). Правило 1: Унификация невозможна (функторы list_create и list_create_help не равны).	Возврат к ТЦ, метка переносится ниже.
2	list_create([], 0, Answer).	ТЦ = list_create([], 0, Answer). Правило 2: Унификация невозможна (функторы list_create и list_create_help не равны).	Возврат к ТЦ, метка переносится ниже.
3	list_create([], 0, Answer).	ТЦ = list_create([], 0, Answer). Правило 3: Унификация невозможна (функторы list_create и list_create_help не равны).	Возврат к ТЦ, метка переносится ниже.
4	list_create([], 0, Answer).	ТЦ = list_create([], 0, Answer). Правило 4: [] = List, 0 = Number, Answer = Answer успех (подобрано знание) => {List = [], Number = 0, Answer = Answer}.	Проверка тела правила 4: list_create_help([], 0, [], Answer).
5	list_create_help([], 0, [], Answer).	ТЦ = list_create_help([], 0, [], Answer). Поиск знания с начала базы знаний. Правило 1: [] = [], Help = [], Help = Answer успех (подобрано знание) => {[] = [], Help = [], Help = Answer}.	Пустое тело заменяет цель в резольвенте.

6	Пусто.	успех - ответ - «Answer = []», метка на правиле 1.	Отказ от найденного значения (откат), возврат к предыдущему состоянию резольвенты.
7	list_create_help([], 0, [], Answer).	ТЦ = list_create_help([], 0, [], Answer). Правило 2: [] = [H|T] унификация невозможна.	Возврат к ТЦ, метка переносится ниже.
8	list_create_help([], 0, [], Answer).	ТЦ = list_create_help([], 0, [], Answer). Правило 3: [] = [_|T] унификация невозможна.	Возврат к ТЦ, метка переносится ниже.
9	list_create_help([], 0, [], Answer).	ТЦ = list_create_help([], 0, [], Answer). Правило 4: Унификация невозможна (функторы list_create_help и list_create не равны).	Отказ от найденного значения (откат), возврат к предыдущему состоянию резольвенты - list_create([], 0, Answer). Метка в конце процедуры - других альтернатив нет => система завершает работу с единственным результатом – «Answer = []».

Реализация программы: сформировать список из элементов, стоящих на нечетных позициях исходного списка (нумерация от 0).

```
domains

        lstI = integer*
        number = integer
predicates

        odd_list_help(lstI, lstI, lstI)
        odd_list(lstI, lstI)
clauses

        odd_list_help([], Help, Help).

        odd_list_help([_|[]], Help, Help) :-!.

        odd_list_help([_|T], Help, Answer) :-
        T = [Next_H|Next_T], New_help = [Next_H|Help],
        odd_list_help(Next_T, New_help, Answer).

        odd_list(List, Answer) :- odd_list_help(List, [], Answer).
```

Порядок и особенности выполнения программы и формирования результата.

В разделе domains необходимо объявить тип списка (integer*).

Чтобы организовать хвостовую рекурсию, необходимо ввести дополнительный параметр Help, который будет накапливать в себе результат (в данном примере - список из элементов, стоящих на нечетных позициях в исходном списке), а при выходе из рекурсии выполнить подстановку: Answer = Help.

Выход из рекурсии будет осуществляться с помошью правила odd_list_help([], _, Help, Help), когда список будет пуст.

Рекурсию организует следующее правило: odd_list_help([_|T], Help, Answer).

На очередном этапе рекурсии происходит выделение второго элемента списка и добавление его в голову результирующего списка (в New_help), а затем использование знания odd_list_help со следующими параметрами: измененным Help - New_help и хвостом хвоста списка.

Чтобы быть уверенным, что получится из списка выделить 2 элемента под-

ряд, присутствует проверка на это: odd_list_help([_|[]], Help, Help), которое будет выполнено, если в списке присутствует только один элемент.

Стоит отметить, что поскольку элементы последовательно добавляются в голову результирующего списка, полученный результат будет перевернутый. Однако, в данном случае нас это не интересует, поскольку в задании заранее не обговаривался порядок, в котором должен был быть сформирован список.

Тесты

Пример 1:

```
goal
        odd_list([1,2,3,4,5,6,7,8,9], Answer).
%Вывод:
        Answer=[8,6,4,2]
        1 Solution
```

Пример 2:

```
goal
        odd_list([1,2,3,4], Answer).
%Вывод:
        Answer=[4,2]
        1 Solution
```

Пример 3:

```
goal
        odd_list([1], Answer).
%Вывод:
        Answer=[]
        1 Solution
```

Пример 4:

```
goal
        odd_list([], Answer).
%Вывод:
        Answer=[]
        1 Solution
```

Реализация программы: удалить заданный элемент из списка (один или все вхождения).

```
domains

        lstI = integer*
        number = integer
predicates

        reverse_help(lstI, lstI, lstI)
        reverse(lstI, lstI)
        delete_elem_from_list_help(lstI, number, lstI, lstI)
        delete_elem_from_list(lstI, number, lstI)
clauses

        reverse_help([], Help, Help).

        reverse_help([H|T], Help, Answer) :-
        reverse_help(T, [H|Help], Answer).

        reverse(List, Answer) :- reverse_help(List, [], Answer).

        delete_elem_from_list_help([], _, Help, Answer) :-
        reverse(Help, Answer).

        delete_elem_from_list_help([H|T], Number, Help, Answer) :-
        H <> Number, !,
        delete_elem_from_list_help(T, Number, [H|Help], Answer).

        delete_elem_from_list_help([_|T], Number, Help, Answer) :-
        delete_elem_from_list_help(T, Number, Help, Answer).

        delete_elem_from_list(List, Number, Answer) :-
        delete_elem_from_list_help(List, Number, [], Answer).
```

Порядок и особенности выполнения программы и формирования результата.

В разделе domains необходимо объявить тип списка (integer*).

Чтобы организовать хвостовую рекурсию, необходимо ввести дополнительный параметр Help, который будет накапливать в себе результат (в данном

примере - список из элементов, не равных заданному числу Number), а при выходе из рекурсии выполнить подстановку: Answer = Help.

Выход из рекурсии будет осуществляться с помошью правила delete_elem_from_li
_, Help, Answer), когда список будет пуст.

Рекурсию организуют следующие два правила: delete_elem_from_list_help([H|T], Number, Help, Answer) и delete_elem_from_list_help([_|T], Number, Help, Answer).

Первое из них выполняется в том случае, если очередной элемент списка не равен заданному значению (H <> Number), при этом второе правило не будет выполняться из-за предиката отсечения !.

И, соответственно, второе правило выполнится, когда первое не будет выполняться - H = Number. Таким образом, элементы списка, равные заданному значению, и элементы, неравные заданному значению, обрабатываются поразному (разными правилами).

Первое вставляет текущий элемент в голову результирующего списка, а второе - игнорирует его.

Стоит отметить, что поскольку элементы последовательно добавляются в голову результирующего списка, полученный результат будет перевернутый. Поэтому, в правиле, предназначенном для выхода из рекурсии вызывается правило reverse(Help, Answer), которое переворачивает список.

Тесты

Пример 1:

```
goal
        delete_elem_from_list([1,2,3,4,5,6,6,7,8,9], 6, Answer).
%Вывод:
        Answer=[1,2,3,4,5,7,8,9]
        1 Solution
```

Пример 2:

```
goal
        delete_elem_from_list([1,2,3,4,5,7,8,9], 6, Answer).
%Вывод:
        Answer=[1,2,3,4,5,7,8,9]
        1 Solution
```

Пример 3:

```
goal
        delete_elem_from_list([1, 1, 1], 1, Answer).
%Вывод:
        Answer=[]
        1 Solution
```

Пример 4:

```
goal
        delete_elem_from_list([], 0, Answer).
%Вывод:
        Answer=[]
        1 Solution
```

Реализация программы: преобразовать список в множество (можно использовать ранее разработанные процедуры).

```
domains

lstI = integer*
number = integer
predicates

        convert_list_to_set_help(lstI, lstI, lstI).
        convert_list_to_set(lstI, lstI).
clauses

        convert_list_to_set_help([], Help, Help).

        convert_list_to_set_help([H|T], Help, Answer) :-
        delete_elem_from_list(T, H, New_T),
        convert_list_to_set_help(New_T, [H|Help], Answer).

        convert_list_to_set(List, Answer) :-
        convert_list_to_set_help(List, [], Answer).
```

Порядок и особенности выполнения программы и формирования результата.

В разделе domains необходимо объявить тип списка (integer*).

Чтобы организовать хвостовую рекурсию, необходимо ввести дополнительный параметр Help, который будет накапливать в себе результат (в данном примере - множество элементов), а при выходе из рекурсии выполнить подстановку: Answer = Help.

Выход из рекурсии будет осуществляться с помошью правила convert_list_to_set_ Help, Help), когда список будет пуст.

Рекурсию организует следующее правило: convert_list_to_set_help([H|T], Help, Answer).

На очередном этапе рекурсии происходит выделение головы списка, удаление элементов, равных голове, из хвоста списка с помощью процедуры delete_elem_fro а затем использование знания convert_list_to_set_help со следующими параметрами: новым хвостом New_T и списком [H|Help].

Стоит отметить, что поскольку элементы последовательно добавляются в голову результирующего списка, полученный результат будет перевернутый. Однако, в данном случае нас это не интересует, поскольку в множестве порядок элементов не важен.

Тесты

Пример 1:

```
goal
        convert_list_to_set([1,2,3,1,2,3,1,2,2,3,4,5,6], Answer).
%Вывод:
        Answer=[6,5,4,3,2,1]
        1 Solution
```

Пример 2:

```
goal
        convert_list_to_set([1,1,1,1,1,1], Answer).
%Вывод:
        Answer=[1]
        1 Solution
```

Пример 3:

```
goal
        convert_list_to_set([], Answer).
%Вывод:
        Answer=[]
        1 Solution
```

Выводы

Эффективность работы каждой программы достигается путем использования хвостовой рекурсии, вводя дополнительные параметры, накапливающие в себе результат.

Ответы на вопросы

1. Как организуется хвостовая рекурсия в Prolog?

Хвостовая рекурсия в Prolog - ссылка на знание, эту же процедуру, последняя в теле правила.

Пример оформления хвостовой рекурсии: p(ar1, ..., argN) :- <выход из рекурсии>. ... p :- t1, ...,tk, p(arg11, ..., argN1).

2. Какое первое состояние резольвенты?

Если задан простой вопрос, то сначала он попадает в резольвенту.

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. 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. Остаток это всегда один (простой или составной) терм.

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

Первые 2 подряд идущих элемента списка: Lst = [First, Second | _], где First, Second - два первых элемента списка Lst.

1-ый и 3- элемент списка: Lst = [First, _, Third|_], где First, Second - 1-ый и 3-ий элементы списка Lst.

5. Как формируется новое состояние резольвенты?

Новая резольвента образуется в два этапа:

1. в текущей резольвенте выбирается одна из подцелей (по стековому принципу - верхняя) и для неё выполняется редукция - замена подцели на тело найденного (подобранного, если удалось) правила,
2. затем, к полученной конъюнкции целей применяется подстановка, полученная как наибольший общий унификатор цели (выбранной) и заголовка сопоставленного с ней правила.

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

Если все метки достигли конца БЗ, то использованы все знания БЗ – и следует остановка работы системы.

Для обозначения точки возврата, система помечает выбранное ранее правило, что позволяет ей перейти к данной метке, и использовать другое, следующее из ниже лежащих знаний в БЗ (не делать повторов). И т.к. БЗ просматривается сверху вниз, то если будет выбрано другое знание для доказательства исходной цели, то метка будет перенесена ниже по тексту БЗ. Т.о. метки ползут вниз. Во время работы системы, некоторое правило может быть использовано несколько раз, тогда оно будет помечено несколькими метками. Первой смещается последняя поставленная метка.