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 №19

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

Москва, 2020 г.

Цель работы

Изучить способы организации, представления и обработки списков в программах на Prolog, методы создания эффективных рекурсивных программ обработки списков и порядок их реализации.

Задачи работы

Приобрести навыки использования 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. Что такое рекурсия? Как организуется хвостовая рекурсия в Prolog? Как можно организовать выход из рекурсии в Prolog?
2. Какое первое состояние резольвенты?
3. В каких пределах программы переменные уникальны?
4. В какой момент, и каким способом системе удается получить доступ к голове списка?
5. Каково назначение использования алгоритма унификации?
6. Каков результат работы алгоритма унификации?
7. Как формируется новое состояние резольвенты?
8. Как применяется подстановка, полученная с помощью алгоритма унификации – как глубоко?
9. В каких случаях запускается механизм отката?
10. Когда останавливается работа системы? Как это определяется на формальном уровне?

Используя хвостовую рекурсию, разработать эффективную программу, (комментируя назначение аргументов), позволяющую:

1. Найти длину списка (по верхнему уровню);
2. Найти сумму элементов числового списка
3. Найти сумму элементов числового списка, стоящих на нечетных позициях исходного списка (нумерация от 0)

Убедиться в правильности результатов

Для одного из вариантов **ВОПРОСА** и одного из **заданий 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 дальнейшие действия – и почему.

Текст процедуры, Вопрос:...

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

Реализация программы: найти длину списка (по верхнему уровню)

```
domains

        lst = integer*
        count, sum = integer
predicates

        len_list(lst, count).
        len_list_help(lst, count Help, count Result).
clauses

        len_list_help([], Help, Help).
        len_list_help([_|T], Help, Answer) :-
        Next_help = Help + 1, len_list_help(T, Next_help, Answer).

        len_list(List, Answer) :- len_list_help(List, 0, Answer).
```

Порядок и особенности выполнения программы и формирования результата.

В разделе domains необходимо объявить тип списка. Было решено сделать список целых чисел (integer*).

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

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

На очередном этапе рекурсии происходит увеличение Help на единицу и использование знания len_list_help в очередной раз с уже новыми параметрами: измененным Help и хвостом списка.

Тесты

Пример 1:

```
goal
        len_list([1, 2], Answer).
```

```
%Вывод:
        Answer=2
        1 Solution
```

Пример 2:

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

Пример 3:

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

Порядок работы системы:

№ шага	Текущая резольвента - ТР	ТЦ, выбираемые правила: сравниваемые термы, подстановка	Дальнейшие действия с комментариями
1	len_list([], Answer).	ТЦ = len_list([], Answer). Правило 1: Унификация невозможна (функторы len_list и len_list_help не равны).	Возврат к ТЦ, метка переносится ниже.
2	len_list([], Answer).	ТЦ = len_list([], Answer). Правило 2: Унификация невозможна (функторы len_list и len_list_help не равны).	Возврат к ТЦ, метка переносится ниже.

3	len_list([], Answer).	ТЦ = len_list([], Answer). Правило 3: [] = List, Answer = Answer успех (подобрано знание) => {List = [], Answer = Answer}.	Проверка тела правила 3: len_list_help([], 0, Answer).
4	len_list_help([], 0, Answer).	ТЦ = len_list_help([], 0, Answer). Поиск знания с начала базы знаний. Правило 1: [] = [], 0 = Help, Answer = Help успех (подобрано знание) => {[] = [], Help = 0, Answer = 0}.	Пустое тело заменяет цель в резольвенте.
5	Пусто	успех - ответ - «Answer = 0», метка на Правиле 1.	Отказ от найденного значения (откат), возврат к предыдущему состоянию резольвенты.
6	len_list_help([], 0, Answer).	ТЦ = len_list_help([], 0, Answer). Правило 2: [] = [_|T] унификация невозможна.	Возврат к ТЦ, метка переносится ниже.
7	len_list_help([], 0, Answer).	ТЦ = len_list_help([], 0, Answer). Правило 3: Унификация невозможна (функторы len_list_help и len_list не равны) => неудача.	Отказ от найденного значения (откат), возврат к предыдущему состоянию резольвенты - len_list([], Answer). Метка в конце процедуры – других альтернатив нет => система завершает работу с единственным результатом – «Answer = 0».

Реализация программы: найти сумму элементов числового списка

```
domains

lst = integer*
count, sum = integer
predicates

sum_list(lst, sum).
sum_list_help(lst, sum Help, sum Answer).
clauses

sum_list_help([], Help, Help).
sum_list_help([H|T], Help, Answer) :-
Next_help = Help + H, sum_list_help(T, Next_help, Answer).

sum_list(List, Answer) :- sum_list_help(List, 0, 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 - сумму элементов списка), а при выходе из рекурсии выполнить подстановку: Answer = Help.

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

На очередном этапе рекурсии происходит увеличение Help на H (голову списка) и использование знания sum_list_help в очередной раз с уже новыми параметрами: измененным Help и хвостом списка.

Тесты

Пример 1:

```
goal
        sum_list([1, 2, 0], Answer).
%Вывод:
```

```
        Answer=3
        1 Solution
```

Пример 2:

```
goal
        sum_list([5], Answer).
%Вывод:
        Answer=5
        1 Solution
```

Пример 3:

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

Реализация программы: найти сумму элементов числового списка, стоящих на нечетных позициях исходного списка (нумерация от 0)

```
domains

lst = integer*
count, sum = integer
predicates

sum_odd_list(lst, sum).
sum_odd_list_help(lst, sum Help, sum Answer).
clauses

sum_odd_list_help([], Help, Help).

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

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

sum_odd_list(List, Answer) :- sum_odd_list_help(List, 0, 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 - сумму нечетных элементов списка), а при выходе из рекурсии выполнить подстановку: Answer = Help.

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

На очередном этапе рекурсии происходит увеличение Help на Next_H (голову хвоста списка) и использование знания sum_odd_list_help в очередной раз с уже новыми параметрами: измененным Help и хвостом хвоста списка.

Чтобы быть уверенным, что получится из списка выделить 2 элемента подряд, присутствует проверка на это: sum_odd_list_help([_|[]], Help, Help), которое будет выполнено, если в списке присутствует только один элемент.

Тесты

Пример 1:

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

Пример 2:

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

Пример 3:

```
goal
        sum_odd_list([4], Answer).
%Вывод:
        Answer=0
        1 Solution
```

Пример 4:

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

Выводы

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

Ответы на вопросы

1. Что такое рекурсия? Как организуется хвостовая рекурсия в Prolog? Как можно организовать выход из рекурсии в Prolog?

Рекурсия – это ссылка при описании объекта на описываемый объект.

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).

При организации выхода из рекурсии необходимо учитываьб, что система использует механизм отката, Следовательно требуется обеспечить, чтобы после выхода из рекурсии система не пробовала использовать вновь ниже лежащие правила (возможно используя отсечение).

2. Какое первое состояние резольвенты?

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

3. В каких пределах программы переменные уникальны?

Переменные уникальны в пределах предложения.

Исключение – анонимные переменные – каждая такая переменная является отдельной сущностью и применяется, когда ее значение неважно для данного предложения.

4. В какой момент, и каким способом системе удается получить доступ к голове списка?

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.

Если начало состоит из одного элемента, то получим: голову и хвост.

5. Каково назначение использования алгоритма унификации?

Назначение алгоритма унификации заключается в попарном сопоставлении термов и попытке построить для них общий пример.

6. Каков результат работы алгоритма унификации?

Результатом использования алгоритма унификации может быть успех или тупиковая ситуация (неудача).

7. 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.

8. Как применяется подстановка, полученная с помощью алгоритма унификации – как глубоко?

Применение подстановки $x_1 = t_1, \ldots, x_n = t_n$ заключается в замене каждого вхождения переменной x_i на соответствующий терм t_i.

9. В каких случаях запускается механизм отката?

Механизм отката запускается, если возникла тупиковая ситуация (достигнут конец БЗ) либо резольвента пуста. В таких случаях происходит откат к предыдущему состоянию резольвенты.

10. 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.