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

О т ч е т

по лабораторной работе № 19

Дисциплина: «Функциональное и логическое программирование»

Выполнила: Овчинникова А. П.

Группа: ИУ7-65Б

Преподаватель: Толпинская Н. Б.

Строганов Ю. В.

Москва, 2020

Задание

Ответить на вопросы (коротко):

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.

Теоретическая часть

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

Рекурсия – это ссылка на определяемый объект во время его определения.

Для осуществления хвостовой рекурсии в Prolog рекурсивный вызов определяемого предиката должен быть последней подцелью в теле рекурсивного правила и к моменту рекурсивного вызова не должно остаться точек возврата. То есть у подцелей, расположенных левее рекурсивного вызова определяемого предиката, не должно оставаться каких-то непроверенных вариантов и у процедуры не должно быть предложений, расположенных ниже рекурсивного правила. Пролог распознает хвостовую рекурсию и устраняет

связанные с ней дополнительные расходы. Этот процесс называется оптимизацией хвостовой рекурсии или оптимизацией последнего вызова.

Базис рекурсии — это предложение, определяющее некую начальную ситуацию или ситуацию в момент прекращения. Как правило, в этом предложении записывается некий простейший случай, при котором ответ получается сразу даже без использования рекурсии. Это предложение часто содержит условие, при выполнении которого происходит выход из рекурсии или отсечение.

2. Какое первое состояние резольвенты?

Первое состояние резольвенты – заданный вопрос.

3. В каких пределах программы переменные уникальны?

Именованные переменные уникальны в рамках предложения, а анонимная переменная – любая уникальна.

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

Доступ к голове списка можно получить во время унификации.

В списковых структурах переменными могут быть представлены и голова, и хвост: [H | T]. Здесь: H – голова, а T – хвост.

5. Каково назначение использования алгоритма унификации?

Процесс унификации запускается автоматически, если есть что доказывать, то надо запускать алгоритм унификации. Пользователь имеет право запустить этот процесс вручную, с помощью утверждения T1 = T2, включенного в текст программы. Если резольвента не пуста – запускается алгоритм унификации (для хранения резольвенты используется стек, соответственно, если стек не пуст – запускается алгоритм унификации).

Назначение унификации – подобрать нужное в данный момент правило. Система подбирает сопоставимые с целью правила с помощью алгоритма унификации.

6. Каков результат работы алгоритма унификации?

В результате унификации формируются подстановки.

7. Как формируется новое состояние резольвенты?

Преобразования резольвенты выполняются с помощью редукции. Новая резольвента образуется в два этапа:

- В текущей резольвенте выбирается одна из подцелей (по стековому принципу – верхняя) и для нее выполняется редукция – замена подцели на тело найденного правила (если удалось найти правило).
- Затем к полученной конъюнкции целей применяется подстановка, полученная как наибольший общий унификатор выбранной цели и заголовка сопоставленного с ней правила.

Если для редукции цели из резольвенты был выбран факт из БЗ, то новая резольвента будет содержать в конъюнкции на одну цель меньше. Если задан простой вопрос и подобран для редукции факт, то произойдет немедленное его согласование. А если для простого вопроса подобрано правило, то число целей в резольвенте не уменьшится, т. к. цель будет заменена телом подобранного правила.

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

В случае успешного согласования программы (базы знаний) и вопроса, в качестве побочного эффекта формируется подстановка, которая содержит значения переменных, при которых вопрос является примером программы. Соответствующие переменные конкретизируются полученными значениями.

9. В каких случаях запускается механизм отката?

В ситуации, когда решение не найдено, и из данного состояния невозможен переход в новое состояние, автоматически включается бэктрэкинг. Происходит возврат к моменту, где еще можно сделать другой альтернативный выбор, то есть к предыдущему состоянию резольвенты. Бэктрэкинг возможен только при наличии альтернативных путей унификации цели.

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

Работа системы останавливается, когда найдены все возможные варианты ответа на вопрос.

Успешное завершение работы программы достигается тогда, когда резольвента пуста.

Если резольвента не пуста и нет утверждений в базе знаний, которые удается сопоставить с выделенной в этой вершине подцелью, то работа системы завершается неудачей.

Код программы

Программа 1. Длина списка.

```
domains

%listI = integer*.
list = integer*.

predicates

list_len(list, integer). % list, result len
list_len_inner(list, integer, integer). % list, inner var, result len

clauses

list_len(List, Result) :- list_len_inner(List, 0, Result).
list_len_inner([], Len, Len) :- !.
list_len_inner([_|T], InnerLen, TotalLen) :-
    NewLen = InnerLen + 1,
    list_len_inner(T, NewLen, TotalLen).
```

```
goal

%list_len([], RES).
%list_len([1], RES).
%list_len([1, 2], RES).
list_len([1, 2, 3], RES).
```

Программа 2. Сумма элементов числового списка.

```
domains

list = integer*.

predicates

list_sum(list, integer). % list, result sum
list_sum_inner(list, integer, integer). %list, inner sum, result sum

clauses

list_sum(List, Result) :- list_sum_inner(List, 0, Result).
list_sum_inner([], Sum, Sum) :- !.
list_sum_inner([H|T], InnerSum, TotalSum) :-
        NewSum = InnerSum + H,
        list_sum_inner(T, NewSum, TotalSum).

goal
```

```
list_sum([1, 2, 3], RES).
%list_sum([1, 1, 1, 1], RES).
%list_sum([6], RES).
%list_sum([0], RES).
%list_sum([1], RES).
%list_sum([], RES).
```

Программа 3. Сумма нечетных элементов числового списка.

```
domains

list = integer*.

predicates

list_sum_odd(list, integer). % list, result sum
list_sum_odd_inner(list, integer, integer). % list, inner sum, result sum

clauses

list_sum_odd(List, Result) :- list_sum_odd_inner(List, 0, Result).
list_sum_odd_inner([], Sum, Sum) :- !.
list_sum_odd_inner([_, H|T], InnerSum, TotalSum) :-
        NewSum = InnerSum + H,
        list_sum_odd_inner(T, NewSum, TotalSum).
list_sum_odd_inner([_|[]], InnerSum, TotalSum) :-
        list_sum_odd_inner([], InnerSum, TotalSum).

goal
```

list_sum_odd([1, 3, 2, 4, 5], RES).

%list_sum_odd([1, 3, 2, 4], RES).

%list_sum_odd([1, 1, 1], RES).

%list_sum_odd([1, 1], RES).

%list_sum_odd([1], RES).

%list_sum_odd([], RES).

Примеры работы программы

Программа 1. Длина списка.

Вопрос	Ответ
list_len([], RES).	RES=0 1 Solution
list_len([1], RES).	RES=1 1 Solution
list_len([1, 2], RES).	RES=2 1 Solution
list_len([1, 2], RES).	RES=3 1 Solution

Программа 2. Сумма элементов числового списка.

Вопрос	Ответ
list_sum([1, 2, 3], RES).	RES=6 1 Solution
list_sum([1, 1, 1, 1], RES).	RES=4 1 Solution
list_sum([6], RES).	RES=6 1 Solution
list_sum([0], RES).	RES=0 1 Solution
list_sum([1], RES).	RES=1 1 Solution
list_sum([], RES).	RES=0 1 Solution

Программа 3. Сумма нечетных элементов числового списка.

Вопрос	Ответ
list_sum_odd([1, 3, 2, 4, 5], RES).	RES=7 1 Solution
list_sum_odd([1, 3, 2, 4], RES).	RES=7 1 Solution

list_sum_odd([1, 1, 1], RES).	RES=1 1 Solution
list_sum_odd([1, 1], RES).	RES=1 1 Solution
list_sum_odd([1], RES).	RES=0 1 Solution
list_sum_odd([], RES).	RES=0 1 Solution

Порядок работы системы для вопроса *list_len([1], RES).* приведен в таблице 1.

Таблица 1. Порядок работы системы.

№ шага	Текущая резольвента – ТР	ТЦ, выбираемые правила: сравниваемые термы, подстановка	Дальнейшие действия с комментариями
1	list_len([1], RES).	Попытка унификации: T1= list_len([1], RES). T2= list_len(List, Result) :- list_len_inner(List, 0, Result). Результат: успех. Подстановка: {List=[1], Result=RES}	Прямой ход. Содержимое резольвенты заменяется телом найденного правила. К резольвенте применяется подстановка.
2	list_len_inner([1], 0, RES).	Попытка унификации: T1= list_len_inner([1], 0, RES). T2= list_len(List, Result) :- list_len_inner(List, 0, Result). Результат: неудача. Разные функторы.	Откат, переход к следующему предложению
3	list_len_inner([1], 0, RES).	Попытка унификации: T1= list_len_inner([1], 0, RES). T2= list_len_inner([], Len, Len) :- !. Результат: неудача, [1] != []	Откат, переход к следующему предложению
4	list_len_inner([1], 0, RES).	Попытка унификации: T1= list_len_inner([1], 0, RES). T2= list_len_inner([_|T], InnerLen, TotalLen) :- NewLen = InnerLen + 1, list_len_inner(T, NewLen, TotalLen). Результат: успех. Подстановка: {_=1, T=[],	Прямой ход. Содержимое резольвенты заменяется телом найденного правила. К резольвенте применяется подстановка.

		InnerLen=0, TotalLen=Res}	
5	NewLen = 0 + 1, list_len_inner([], NewLen, Res).	ТЦ: NewLen = 0 + 1, Результат: успех. Подстановка: {NewLen=1}	Прямой ход. К резольвенте применяется подстановка.
6	list_len_inner([], 1, Res).	Попытка унификации: T1= list_len_inner([], 1, Res). T2= list_len_inner([], 1, Res). Результат: неудача. Разные функторы.	Откат, переход к следующему предложению
7	list_len_inner([], 1, Res).	Попытка унификации: T1= list_len_inner([], 1, Res). T2= list_len_inner([], Len, Len) :- !. Результат: успех. Подстановка: {[]=[], Len=1, Len=Res}	Прямой ход. Содержимое резольвенты заменяется телом найденного правила. К резольвенте применяется подстановка.
8	!.	Выполнение отсечения. !. Результат: успех.	Резольвента становится пустой.
9	Пусто	Все переменные связанны.	Найдено одно решение. Вывод найденного решения на экран. Попытка отката завершает использование процедуры. Система завершает работу.

Вывод

Таким образом, в программах эффективность достигается за счет использования хвостовой рекурсии и отсечения.