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»

ОТЧЕТ

к лабораторной работе № 19

По курсу: «Функциональное и логическое программирование»

Студент ИУ7-64Б
Лозовский А.А.

Преподаватель
Толпинская Н.Б

Москва, 2020 г.

Задание

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.

Ответы на вопросы

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

Рекурсия – один из способов организации повторых вычислений. В логическом программировании – способ заставить систему многократно использовать одну и ту же процедуру. При этом из нее должен быть выход.

Организация хвостовой рекурсии:

- Рекурсивный вызов единственен и расположен в конце тела правила.
- До вычисления рекурсивного вызова не должно быть возможности сделать откат (т.е точки отката отсутствуют). Этого можно добиться, например, с помощью предиката отсечения.

Для выхода из рекурсии используется отдельное правило, в конце которого будет находиться предикат отсечения.

2. Какое первое состояние резольвенты?

На первом шаге в резольвенте находится заданный вопрос (цель).

3. В каких пределах программы переменные уникальны?

Именованная переменная уникальна в рамках предложения, в котором она используются. Любая анонимная переменная является уникальной.

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

Во время выполнения алгоритма унификации системе удается получить доступ к голове списка. Во время унификации система пытается разделить список на «начало» и «конец», чтобы унификация была успешна.

5. Каково назначение использования алгоритма унификации?

Для поиска ответа на вопрос системе необходимо найти подходящее знание в БЗ, для поиска такого знания используется алгоритм унификации. Формально, он помогает системе понять, что заголовок подошел: алгоритм попарно пытается сопоставить термы (текущую цель и термы из БЗ) и построить для них общий пример (для этого используется подстановка).

6. Каков результат работы алгоритма унификации?

Алгоритм унификации может завершиться «успехом» и «неудачей». В случае успеха результирующая ячейка будет содержать подстановку(наиболее общий унификатор).

7. Как формируется новое состояние резольвенты?

Преобразования резольвенты выполняются с помощью редукции – замены текущей цели на тело найденного в программе правила (с помощью унификации текущей цели и заголовка правила программы).
Преобразование резольвенты разделено на два этапа:

1) Берется верхняя из подцелей резольвенты (по стековому принципу) и заменяется на тело правила, найденного в программе.
2) Затем к полученной конъюнкции целей применяется подстановка (наибольший общий унификатор цели и сопоставленного с ней правила).

8. **Как применяется подстановка, полученная с помощью алгоритма унификации – как глубоко?**
Подстановкой называется множество пар, вида: $\{ X_i = t_i \}$, где X_i – переменная, а t_i – терм. Т.е происходит конкретизация переменной термом. Применение подстановки заключается в замене каждого вхождения переменной X_i на соответствующий терм (t_i). В результате применения подстановки переменные конкретизируются значениями, которые будут далее использованы при доказательстве истинности тела выбранного правила то есть значения переменных переходят на следующих шаг доказательства.

9. **В каких случаях запускается механизм отката?**
Во время работы системы, в случае, если решение не найдено, и из данного состояния невозможен переход в новое состояние (тупиковое состояние), применяется механизм отката. Также для поиска альтернативных решений (резольвента пуста, но не все правила были рассмотрены).

10. **Когда останавливается работа системы? Как это определяется на формальном уровне?**
Система завершает работу, в случае если метка расположна в конце процедуры(которая доказывается для ответа на поставленный вопрос) и не осталось альтернатив (для каждой подцели были найдены все возможные наборы значений и везде были проставлены метки), либо если ответ не был найден, но были просмотрены все возможные варианты.

```
domains
    list = integer*

predicates
    len(list, integer)
    len(list, integer, integer)

    sum(list, integer)
    sum(list, integer, integer)

    sum_odd(list, integer)
    sum_odd(list, integer, integer)

clauses
    /*Задание 1*/
    len(List, Len) :- len(List, 0, Len).
    len([], Len, Len) :-!.
    len([_|T], TmpLen, Len) :- NewLen = TmpLen + 1,
                               len(T, NewLen, Len).


    /* Задание 2*/
    sum(List, Sum) :- sum(List, 0, Sum).
    sum([], Sum, Sum) :-!.
    sum([H|T], TmpSum, Sum) :- NewSum = TmpSum + H,
                               sum(T, NewSum, Sum).
```

```
/* Задание 3*/
sum_odd(List, Sum) :- sum_odd(List, 0, Sum).
sum_odd([], Sum, Sum) :-!.

/*если в списке последний элемент на четной позиции*/
sum_odd([_|[]], TmpSum, Sum)  :- sum_odd([], TmpSum, Sum).

/*обработка каждого нечетного элемента списка*/
sum_odd([_,H|T], TmpSum, Sum) :- NewSum = TmpSum + H,
                                 sum_odd(T, NewSum, Sum).
```

Описание аргументов

1) len(list, integer)
   - первый аргумент – список, который необходимо обработать
   - второй аргумент – результат (длина списка)

2) len(list, integer, integer)
   - первый аргумент – список, который необходимо обработать
   - второй аргумент – текущая длина (отвечает за «накапливание» результата)
   - третий аргумент – результат, когда длина будет найдена, будет связана с полученным значением.
3) sum(list, integer)
   - первый аргумент – список, который необходимо обработать
   - второй аргумент – результат (сумма всех элементов списка)

4) sum(list, integer, integer)
   - первый аргумент – список, который необходимо обработать
   - второй аргумент – текущая сумма (отвечает за «накапливание» результата)
   - третий аргумент – результат, когда сумма будет найдена, будет связана с полученным значением.

5) sum_odd(list, integer)
   - первый аргумент – список, который необходимо обработать
   - второй аргумент – результат (сумма всех элементов на нечетной позиции списка)

6) sum_odd(list, integer, integer)
   - первый аргумент – список, который необходимо обработать
   - второй аргумент – текущая сумма (отвечает за «накапливание» результата)
   - третий аргумент – результат, когда сумма будет найдена, будет связана с полученным значением.

Примеры целей и результатов работы программы

1. **Goal** len([], Len).
   Result Len=0

2. **Goal** len([1, 2, 3, 4], Res).

Result Res=4

3. **Goal** sum([1, 2, 3], Sum).
   Result Sum=6

4. **Goal** sum_odd([1, 2, 3, 4], Sum).
   Result Sum=6

5. **Goal** sum_odd([1, 2, 3], Sum).
   Result Sum=2

6. **Goal** sum_odd([1], Sum).
   Result Sum=0

Описание порядка поиска объектов

В таблице будет отображен только проход по процедурам, текст которых приведен ниже, остальные шаги опущены (термы будут не унифицируемы из-за разных имен главных функторов).

Текст процедуры

(две процедуры sum(list, integer), sum(list, integer, integer)**):**

Пронумеруем правила:

```
I   sum(List, Sum) :- sum(List, 0, Sum).
II  sum([], Sum, Sum) :- !.
III sum([H|T], TmpSum, Sum) :- NewSum = TmpSum + H,
                               sum(T, NewSum, Sum).
```

Вопрос

Goal sum([1], Res).

№ шага	Текущая резольвента – ТР	ТЦ, выбираемые правила: сравниваемые термы, подстановка	Дальнейшие действия с комментариями
Шаг1	sum([1], Res).	ТЦ: sum([1], Res).	Поиск знания с начала базы знаний.
	sum([1], Res).	ТЦ: sum([1], Res). Сравниваемые термы: sum([1], Res) ПРI: sum(List, Sum)	Проверка тела ПРI.

		Результат: успех (подобрано знание) Подстанока: {List=[1], Res=Sum}	
Шаг2	sum([1], 0, Sum). Резольвента изменилась в 2 этапа.	ТЦ: sum([1], 0, Sum).	Поиск знания с начала базы знаний.
	sum([1], 0, Sum).	ТЦ: sum([1], 0, Sum). Сравниваемые термы: sum([1], 0, Sum) ПРI: sum(List, Sum) Результат: унификация невозможна	Возврат к ТЦ, метка переносится ниже.
	sum([1], 0, Sum).	ТЦ: sum([1], 0, Sum). Сравниваемые термы: sum([1], 0, Sum) ПРII: sum([], Sum, Sum). Результат: унификация невозможна	Возврат к ТЦ, метка переносится ниже.
	sum([1], 0, Sum).	ТЦ: sum([1], 0, Sum). Сравниваемые термы: sum([1], 0, Sum) ПРIII: sum([H|T], TmpSum, Sum) Результат: успех (подобрано знание) Подстановка: {H=1, T=[], TmpSum=0, Sum=Sum}	Проверка тела ПРIII.
Шаг3	NewSum = 0 + 1; sum([], NewSum, Sum) Резольвента изменилась в 2 этапа.	ТЦ: NewSum = 0 + 1 Результат: успех, конкретизация NewSum. Подстановка: { NewSum=1 }	Переход к следующей цели.
Шаг4	sum([], 1, Sum) Резольвента изменилась в 2 этапа.	ТЦ: sum([], 1, Sum)	Поиск знания с начала базы знаний.
	sum([], 1, Sum)	ТЦ: sum([], 1, Sum) Сравниваемые термы: sum([], 1, Sum) ПРI: sum(List, Sum) Результат: унификация невозможна	Возврат к ТЦ, метка переносится ниже.
	sum([], 1, Sum)	ТЦ: sum([], 1, Sum) Сравниваемые термы: sum([], 1, Sum)	Проверка тела ПРII

		ПРII: sum([], Sum, Sum). Результат: успех (подобрано знание) Подстановка: {1=Sum, Sum=1} Sum=Sum – связанные (я не знаю как это правильно отразить в подстановке)	
Шаг5	!	ТЦ: ! Результат: успех, выполнение отсечения.	Пустое тело заменяет цель в резольвенте
	Пусто		Успех, однократный ответ, подобрано ПРII. Ответ: Sum=1 Отказ от найденного значения – при отмете ! - завершение использования процедуры. Все метки в конце процедуры => система завершает работу.

Вывод

Для повышения эффективности программы на пролог, можно использовать отсечения, чтобы ограничить количество вычислений, в случае, если они избыточны (например, как в случае с взаимоисключающими правилами), также можно использовать хвостовую рекурсию, ее главное отличие от стандартной реализации рекурсии в том, что при вычислениях, не возникает истощения памяти.

Исправления

ЛР13

Предложения, не содержащие переменные, называются составными!!! Почему тот же текст?????Не исправили?

Ответ: База знаний состоит из предложений – фактов и правил, причем факт – частный случай правила (отсутствует тело). Также третий вид предложений в пролог – вопросы (состоит только из тела – составного терма).

Пример факта из программы**: automobile**("Surname1", "Ford", "Black", 1600000).

Пример правила из программы**: search_by**(Surname, PhoneNum, CarBrand, CarPrice) :-
phonebook (Surname, PhoneNum, _),
automobile (Surname, CarBrand, _, CarPrice).

Пример вопроса из программы: **search_by**(Surname, "0-000-111-222", CarBrand, CarPrice).

Что такое пример терма? Как и когда строится? Как Вы думаете, система строит и хранит примеры?

Примером терма B называется такой терм A, если существует такая подстановка в терм A, которая в результате будет эквивалентна терму B. НЕТ!!! Что является более общим термом??? **Я думаю, что примеры термов строятся при доказательстве заданной пользователем цели или при вычислении…**

Ответ: Терм B называется примером терма A, если существует такая подстановка Θ, что B=AΘ, где AΘ – результат применения подстановки.

ЛР14

На 7-м шаге табл «Унификация»:

Из стека вытолкнули: *Bank=Bank – стек стал пустым – алгоритм унификации не заканчивает работу???*

Текст программы

```
domains
        adress = adress(symbol City, symbol Street, integer HouseNum, integer FlatNum)

predicates
        phonebook (symbol Surname, symbol PhoneNum, adress Adrr)
        automobile(symbol Surname, symbol Brand, symbol Colour, integer Price)
        investors (symbol Surname, symbol Bank,  symbol Acc_number, integer Value)

        search_by(symbol CarBrand, symbol CarColour, symbol Surname, symbol City, symbol PhoneNum, symbol
Bank)

clauses
        phonebook("Surname1", "0-000-111-222", adress("Moscow", "Unnatov", 14, 128)).
        phonebook("Surname1", "5-666-777-888", adress("St.Peterburg", "Nevskiy", 14, 128)).
        phonebook("Surname2", "1-222-333-444", adress("Moscow", "8th March", 12, 153)).
        phonebook("Surname3", "3-444-555-666", adress("Moscow", "Mishina", 1, 10)).
        phonebook("Surname4", "9-000-111-111", adress("St.Peterburg", "Nevskiy", 1, 10)).

        /*3 drivers, 1 doesn't have investments*/
        automobile("Surname1", "Ford",  "Black",  1600000).
        automobile("Surname2", "Ford",  "Black",  1600000).
        automobile("Surname3", "Ford",  "Black",  1600000).

        /*1 driver*/
        automobile("Surname4", "Volvo",      "Silver", 1300000).

        /*no drivers, doesn't have investments*/
        automobile("Surname3", "Nissan",     "Red",    1300000).

        investors("Surname1", "Sberbank",       "0000 4444 3333 2222", 700000).
        investors("Surname2", "Sberbank",       "0000 2222 3333 2222", 200000).
        investors("Surname4", "MoscowBank", "0000 3333 3333 2222", 300000).
        investors("Surname4", "VTB",            "0000 4444 3333 2222", 700000).

        /*searches for surname, city, phone, bank*/
        search_by(CarBrand, CarColour, Surname, City, PhoneNum, Bank) :-
                        automobile(Surname, CarBrand, CarColour, _),
                        phonebook(Surname, PhoneNum, adress(City, _, _, _)),
                        investors(Surname, Bank, _, _).
```

Вопрос

Goal search_by("Volvo", "Silver", Surname, City, PhoneNum, Bank)

шаг	результирующая ячейка	рабочее поле	пункт Алг.	стек
0			1.	search_by("Volvo", "Silver", Surname, City, PhoneNum, Bank) = search_by(CarBrand, CarColour, Surname, City, PhoneNum, Bank)
1		search_by("Volvo", "Silver", Surname, City, PhoneNum, Bank) = search_by(CarBrand, CarColour, Surname, City, PhoneNum, Bank) ⟶	Е.	CarBrand="Volvo", CarColour="Silver", Surname= Surname, City=City, PhoneNum=PhoneNum, Bank=Bank
2	CarBrand="Volvo"	CarBrand="Volvo" ⟵	Г.	CarColour="Silver", Surname= Surname, City=City, PhoneNum=PhoneNum, Bank=Bank
3	CarBrand="Volvo", CarColour="Silver"	CarColour="Silver" ⟵	Г.	Surname= Surname, City=City, PhoneNum=PhoneNum, Bank=Bank
4	CarBrand="Volvo", CarColour="Silver", Surname= Surname	Surname= Surname ⟵	-	City=City, PhoneNum=PhoneNum, Bank=Bank
5	CarBrand="Volvo", CarColour="Silver", Surname= Surname, City=City	City=City ⟵	-	PhoneNum=PhoneNum, Bank=Bank
6	CarBrand="Volvo", CarColour="Silver", Surname= Surname, City=City, PhoneNum=PhoneNum	PhoneNum=PhoneNum ⟵	-	Bank=Bank
7	CarBrand="Volvo", CarColour="Silver", Surname= Surname, City=City, PhoneNum=PhoneNum, Bank=Bank	Bank=Bank ⟵	-	Пусто
8	Полученная подстановка: CarBrand="Volvo", CarColour="Silver", Surname= Surname, City=City, PhoneNum=PhoneNum, Bank=Bank	Т.к. стек пуст – **успех** и в рез. ячейке подстановка		