|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 ИУ7-65Б

Юмаев А.Р.

Преподаватель

Толпинская Н.Б

Строганов Ю.В.

*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**

* **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.

Одним из способов организации повторых вычислений является рекурсия. Если говорить о логическом программировании, то рекурсией называется способ заставить систему много раз использовать одну и ту же процедуру, так же, для корректности работы у рекурсии должен быть выход.

Организация хвостовой рекурсии:

* Рекурсивный вызов должен быть расположен к конце тела правила
* Рекурсивный вызов должен быть единственным
* До вычисления рекурсивного вызова не должно быть точек отката (этого можно достичь, например, поставив предикат отсечения перед рекурсивным вызовом).

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

* **Какое первое состояние резольвенты?**

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

* **В каких пределах программы переменные уникальны?**

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

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

Во время попытки унификации списка система получает доступ к «началу» списка. Она (система) пытается разделить список на «начало» и «конец», чтобы унификация была успешна.

* **Каково назначение использования алгоритма унификации?**

Алгоритм унификации формализует механизм логического вывода и является основным вычислительным шагом программы. Он предназначен для доказательства истинности (поиска соответствий в БЗ) текущей цели. Системе, для доказательства (поиска ответа на вопрос) цели, необходимо найти знание в БЗ. Алгоритм унификации формализует процесс поиска. Путем сопоставления двух термов (цели из резольвенты и предложений из БЗ) алгоритм помогает системе понять, что заголовок подошел.

* **Каков результат работы алгоритма унификации?**

Алгоритм унификации может завершиться неудачей и успехом, в случае успеха в качестве побочного эффекта будет построен наиболее общий унификатор (подстановка).

* **Как формируется новое состояние резольвенты?**

В резольвенте хранятся цели, истинность которых необходимо доказать. Для хранения целей в резольвенте используется стек. Она (резольвента) меняется в ходе доказательства с помощью алгоритма редукции (замена текущей цели на тело правила, найденного в БЗ с помощью алгоритма унификации). Преобразование происходит следующим образом:

* берется верхняя цель и заменяется на тело правила (если речь идет о факте, так как факт - частный случай правила, не имеющий тела, цель просто убирается), найденного путем сопоставления алгоритмом унификации из БЗ.
* к полученной конъюнкции целей применяется подстановка, полученная алгоритмом унификации
* **Как применяется подстановка, полученная с помощью алгоритма унификации – как глубоко?**

Подстановкой называется множество пар, вида: { = } , где – переменная, а – терм. Т.е происходит конкретизация переменной термом. Применение подстановки заключается в замене каждого вхождения переменной на соответствующий терм (). В результате применения подстановки конкретизированные переменные могут быть использованы для дальнейшего доказательства истинности тела правила.

* **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).

* **Когда останавливается работа системы? Как это определяется на формальном уровне?**

­Система завершает работу, в случае, если были найдены все варианты(метка расположена в конце тела правила, найденного при унификации с заданной пользователем целью) и для всех подцелей и заголовка просмотрены все предложения в БЗ, либо если не было найдено результатов и конец БЗ.

|  |
| --- |
| **domains**   intList = integer\*   **predicates**  length(intList, integer)  my\_len(intList, integer, integer)  count\_all(intList, integer)  my\_count(intList, integer, integer)  count\_odd(intList, integer)  my\_count\_odd(intList, integer, integer)   **clauses**  % задание 1  length(List, Len) :- my\_len(List, 0, Len).  my\_len([], Len, Len) :- !.  my\_len([\_|T], TmpLen, Len) :- NewLen = TmpLen + 1,  my\_len(T, NewLen, Len).    % задание 2  count\_all(List, Sum) :- my\_count(List, 0, Sum).  my\_count([], Sum, Sum) :- !.  my\_count([H|T], TmpSum, Sum) :- NewSum = TmpSum + H,   my\_count(T, NewSum, Sum).     % задание 3  count\_odd(List, Sum) :- my\_count\_odd(List, 0, Sum).  my\_count\_odd([], Sum, Sum) :- !.  my\_count\_odd([\_|[]], TmpSum, Sum) :-  my\_count\_odd([], TmpSum, Sum).  my\_count\_odd([\_,H|T], TmpSum, Sum) :- NewSum = TmpSum + H,   my\_count\_odd(T, NewSum, Sum). |

**Описание программы**

1. **length(intList, integer)**: первый аргумент – обрабатываемый список, второй – результат
2. **my\_len(intList, integer, integer)**: первый аргумент – обрабатываемый список, второй – текущая длина, третий – результат(будет записан при выходе из рекурсии).
3. **count\_all(intList, integer)**: первый аргумент – обрабатываемый список, второй – результат
4. **my\_count(intList, integer, integer)**: первый аргумент – обрабатываемый список, второй – текущая сумма, третий – результат(будет записан при выходе из рекурсии).
5. **count\_odd(intList, integer)**: первый аргумент – обрабатываемый список, второй – результат
6. **my\_count\_odd(intList, integer, integer)**: первый аргумент – обрабатываемый список, второй – текущая сумма, третий – результат(будет записан при выходе из рекурсии).

**Примеры целей и результатов работы программы**

**Goal** length([], Len).

**Result** Len=0

**Goal** count\_all([1, 2], Sum).

**Result** Sum=3

**Goal** count\_odd([1, 2, 3, 4], Sum).

**Result** Sum=10

**Goal** count\_odd([1, 2, 3], Sum).

**Result** Sum=2

**Goal** count\_odd([1], Sum).

**Result** Sum=0

**Описание порядка поиска объектов**

**Текст процедуры (две процедуры – count\_all, my\_count):**

1. count\_all(List, Sum) :- my\_count(List, 0, Sum).
2. my\_count([], Sum, Sum) :- !.
3. my\_count([H|T], TmpSum, Sum) :- NewSum = TmpSum + H,

my\_count(T, NewSum, Sum).

**Вопрос:** count\_all([2], Sum).

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Текущая резольвента – ТР | ТЦ, выбираемые правила: сравниваемые термы,  подстановка | Дальнейшие действия с комментариями |
| ***Шаг1*** | count\_all([2], Sum). | ***ТЦ:***count\_all([2], Sum). | *Поиск знания с начала базы знаний.* |
|  | count\_all([2], Sum). | ***ТЦ*:** count\_all([2], Sum).  **Сравниваниемые термы:**  count\_all([2], Sum)  count\_all([List, Sum) (ПР1)  **Результат:** успех (подобрано знание)  ***Подстанока:*** *{List=[2], Sum=Sum}* | *Проверка тела ПР1.* |
| ***Шаг2*** | my\_count([2], 0, Sum).  Резольвента изменилась в 2 этапа. | ***ТЦ:*** my\_count([2], 0, Sum). | *Поиск знания с начала базы знаний.* |
|  | my\_count([2], 0, Sum). | ***ТЦ:*** my\_count([2], 0, Sum).  **Сравниваниемые термы**:  my\_count([2], 0, Sum)  count\_all (List, Sum) (*ПР1*)  **Результат:** унификация невозможна | *Возврат к ТЦ, метка переносится ниже.* |
|  | my\_count([2], 0, Sum). | ***ТЦ:***my\_count([2], 0, Sum).  ***Сравниваемые термы:***  my\_count([2], 0, Sum)  my\_count([], Sum, Sum). (ПР2)  **Результат:** унификация невозможна | *Возврат к ТЦ, метка переносится ниже.* |
|  | my\_count([2], 0, Sum). | ***ТЦ:***my\_count([2], 0, Sum).  ***Сравниваемые термы:***  my\_count([2], 0, Sum)  my\_count([H|T], TmpSum, Sum) (ПР3)  **Результат:** успех (подобрано знание)  **Подстановка:** {H=1, T=[], TmpSum=0, Sum=Sum} | *Проверка тела ПР3.* |
| ***Шаг3*** | NewSum = 0 + 1; my\_count([], NewSum, Sum)  Резольвента изменилась в 2 этапа. | ***ТЦ:*** NewSum = 0 + 1  **Результат:** успех, конкретизация NewSum.  **Подстановка:** { NewSum=1 } | *Переход к следующей цели.* |
| ***Шаг4*** | my\_count([], 1, Sum)  Резольвента изменилась в 2 этапа. | ***ТЦ:*** my\_count([], 1, Sum) | *Поиск знания с начала базы знаний.* |
|  | my\_count([], 1, Sum) | ***ТЦ:*** my\_count([], 1, Sum)  **Сравниваемые термы:**  my\_count([], 1, Sum)  count\_all (List, Sum) (*ПР1*)  **Результат:** унификация невозможна | *Возврат к ТЦ, метка переносится ниже.* |
|  | my\_count([], 1, Sum) | ***ТЦ:*** my\_count([], 1, Sum)  **Сравниваемые термы:**  my\_count([], 1, Sum)  my\_count([], Sum, Sum). (ПР2)  **Результат:** успех (подобрано знание)  **Подстановка:** {Sum=1} | *Проверка тела ПР2* |
| ***Шаг5*** | ! | ***ТЦ:*** *!*  ***Результат:*** *успех, отсечение.* | *Пустое тело заменяет цель в резольвенте* |
|  | Пусто | ***Успех, однократный ответ, подобрано ПР2.*** | *Ответ: Sum=1 Отказ от найденного значения. Попытка отката ! приводит к завершению испольщования процедуры. Все метки в конце процедуры => система завершает работу.* |

**Выводы**

Для повышения эффетивности программы на пролог, можно использовать следующее:

* Предикат отсечения, с целью избавиться от лишних вычислений, как, например, при поиске максимального/минимального числа (взаимоисключающие правила)
* При использовании отсечения, мне кажется, порядок следования предложений в БЗ также может помочь в повышении эффективности
* Хвостовая рекурсия, которая в отличии от обычной рекурсии, не тратит дополнительно память.