|  |  |
| --- | --- |
| 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»\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Лабораторная работа № 19**

|  |  |
| --- | --- |
| **Тема** Prolog  **Студент** Белоусова Ю.С.  **Группа** ИУ7-61Б  **Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  **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

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

Рекурсия - ссылка на себя. В Prolog это способ повторного использования процедуры.

Хвостовая рекурсия в Prolog организуется за счет расположения повторного вызова функции последней подцелью в конъюнктивном правиле.

В Prolog рекурсия организуется с помощью нескольких правил, часть из которых не являются рекурсивными и служат для выхода из рекурсии, они используют отсечения для выхода из рекурсии.

#### 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.

1. **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.

#### Каково назначение использования алгоритма унификации?

Назначение алгоритма унификации - подобрать знание, которое позволит на поставленный  
 вопрос ответить да.

#### Каков результат работы алгоритма унификации?

Результатом алгоритма унификации является «успех» (цель доказана) или  
 «неудача» (на основе данной базы знаний невозможно доказать цель). В случае «успеха» в  
 качестве побочного эффекта формируется подстановка, содержащая значения переменных, при которых вопрос станет примером программы.

#### 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).

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

Подстановкой называется множество пар, вида: { 𝑋𝑖= 𝑡𝑖 } , где 𝑋𝑖 – переменная, а 𝑡𝑖 – терм. Т.е происходит конкретизация переменной термом. Применение подстановки заключается в замене каждого вхождения переменной 𝑋𝑖 на соответствующий терм (𝑡𝑖). 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.

#### 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).

1. **Когда останавливается работа системы? Как это определяется на формальном уровне?** Когда резольвента пуста и все знания в БЗ отмечены.

## 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).

1. len(list, integer)

**Описание аргументов**

* + первый аргумент – список, который необходимо обработать
  + второй аргумент – результат (длина списка)

1. 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.
2. sum(list, integer)
   * первый аргумент – список, который необходимо обработать
   * второй аргумент – результат (сумма всех элементов списка)
3. 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.
4. sum\_odd(list, integer)
   * первый аргумент – список, который необходимо обработать
   * второй аргумент – результат (сумма всех элементов на нечетной позиции списка)
5. 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

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

**Result** Sum=2

1. **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:

1. sum(List, Sum) :- sum(List, 0, Sum).
2. sum([], Sum, Sum) :- !.
3. 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).  Сравниваниемые термы: sum([1], Res)  ПРI: sum(List, Sum) | Проверка тела ПРI. Изменение резольвенты - замена цели на тело правило. Применение подстановки к резольвенте. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Результат: успех (подобрано знание) Подстанока: {List=[1], Res=Sum} |  |
| Шаг2 | sum([1], 0, Sum). | ТЦ: sum([1], 0, Sum). | Поиск знания с начала базы знаний. |
|  | sum([1], 0, Sum). | ТЦ: sum([1], 0, Sum).  Сравниваниемые 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. Изменение резольвенты - замена цели на тело правило. Применение подстановки к резольвенте. |
| Шаг3 | NewSum = 0 + 1; sum([], NewSum, Sum) | ТЦ: NewSum = 0 + 1  Результат: успех, конкретизация NewSum. Подстановка: { NewSum=1 } | Прямой ход. |
| Шаг4 | sum([], 1, Sum) | 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. Изменение резольвенты - замена цели на тело правило. Применение подстановки к резольвенте. |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | ПРII: sum([], Sum, Sum).  Результат: успех (подобрано знание) Подстановка: {1=Sum, Sum= Sum= 1} |  |
| Шаг5 | ! | ТЦ: !  Результат: успех, выполнение отсечения. | Прямой ход. |
|  | 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 эффетивности 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.