|  |  |
| --- | --- |
| 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**

**по курсу “Функциональное и логическое программирование”**

1. **по теме** “ **Обработка списков на Prolog**”

|  |  |
| --- | --- |
| Студент: | Уласик Е.А. |
| Группа: | ИУ7-61 |
| Преподаватель: | Толпинская Н.Б. |

*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 | … | … | … |
| … | … | … | … |

Вопросы

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 организации хвостовой рекурсии в Prolog рекурсивный вызов определяемого предиката должен быть последней подцелью в теле рекурсивного правила. Выход из рекурсии в Prolog организован с помощью использования отсечения.

1. **Какое первое состояние резольвенты?**

Первое состояние резольвенты – это вопрос.

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

Именованные переменные уникальны в рамках одного предложения, анонимные уникальны все.

1. **В какой момент, и каким способом системе удаётся получить доступ к голове списка?**

В Prolog используется специальный символ вертикальной черты | для разделения списка на голову и хвост. Например, [1, 2, 3] или [1 | [2, 3]]. В правиле pred([H|T]) :- … при унификации H – будет хранить значение головы, а T – хвоста списка.

1. **Каково назначение использования алгоритма унификации?**

Назначение алгоритма унификации в сопоставлении двух термов. В результате унификации может быть успех или неудача. Побочный результат алгоритма унификации является конкретизация переменных.

1. **Каков результат работы алгоритма унификации?**

Алгоритм унификации может завершиться получением одного из двух результатов – успехом или тупиковой ситуацией, то есть неудачей. Если вопрос содержит именованные переменные, то в процессе унификации они конкретизируются и являются результатом работы программы. Программа выведет все варианты конкретизации этих переменных и количество найденных решений.

1. **Как формируется новое состояние резольвенты?**

Резольвента изменяется с использованием редукции, то есть замены подцели телом правила, с которым унифицируется данная подцель, и механизма отката. В ходе отката резольвента возвращается в предыдущее состояние.

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

Если для этих переменные найдено значение, все переменные в резольвенте конкретизируются значениями полученной подстановки. Существуют переменные, имеющие одинаковое имя, но разные области видимости соответственно они являются разными переменными.

1. **В каких случаях запускается механизм отката?**

Механизм отката запускается в случае, если в ходе поиска доказательства подцели программа зашла в тупиковую ситуацию, и, если резольвента не пуста и решение найдено, но в базе знаний остались не отмеченные предложения.

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

Когда резольвента пуста и все знания в БЗ отмечены.

Листинг программы

domains

num = integer.

int\_list = integer\*.

predicates

**len**(int\_list, num).

**len\_inner**(int\_list, num, num).

**sum**(int\_list, num).

**sum\_inner**(int\_list, num, num)

**sum\_odd**(int\_list, num).

**sum\_odd\_inner**(int\_list, num, num).

clauses

**len**(INT\_LIST, RES) :- **len\_inner**(INT\_LIST, RES, **0**).

**len\_inner**([], RES, RES) :- !.

**len\_inner**([**\_**|Tail], RES, CUR\_LEN) :-

CUR\_LEN1 = CUR\_LEN + **1**,

**len\_inner**(Tail, RES, CUR\_LEN1).

**sum**(INT\_LIST, RES) :- **sum\_inner**(INT\_LIST, RES, **0**).

**sum\_inner**([], RES, RES) :- !.

**sum\_inner**([Head|Tail], RES, CUR\_SUM) :-

CUR\_SUM1 = CUR\_SUM + Head,

**sum\_inner**(Tail, RES, CUR\_SUM1).

**sum\_odd**(INT\_LIST, RES) :- **sum\_odd\_inner**(INT\_LIST, RES, **0**).

**sum\_odd\_inner**([], RES, RES) :- !.

**sum\_odd\_inner**([**\_**], RES, RES) :- !.

**sum\_odd\_inner**([**\_**, Num|Tail], RES, SUM) :-

SUM1 = NUM + SUM,

**sum\_odd\_inner**(Tail, RES, SUM1).

goal

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

% sum([2, 4, 1, 7, -12, 3], RES).

**sum\_odd**([**1**, **5**, **2**, **4**, **3**], RES).

Обозначения

1. len –правило-обёртка, которое получает 2 аргумента:

* список, длину которого нужно вычислить;
* число, в которое будет записан результат.

1. len\_inner – привило, в котором осуществляется хвостовая рекурсия и которое получает 3 аргумента:

* список, длину которого нужно вычислить;
* число, в которое будет записан результат;
* число, количество уже пройденных элементов списка.

1. sum – правило-обёртка, которое получает 2 аргумента:

* список, сумму которого нужно вычислить;
* число, в которое будет записан результат.

1. sum\_inner – правило, в котором осуществляется хвостовая рекурсия и которое получает 3 аргумента:

* список, сумму которого нужно вычислить;
* число, в которое будет записан результат.
* число, в которое записывается промежуточная сумма.

1. sum\_odd – правило-обёртка, которое получает 2 аргумента:

* список, сумму элементов на нечётных позициях которого нужно вычислить;
* число, в которое будет записан результат.

1. sum\_odd\_inner – правило, в котором осуществляется хвостовая рекурсия и которое получает 3 аргумента:

* список, сумму элементов на нечётных позициях которого нужно вычислить;
* число, в которое будет записан результат;
* число, в которое записывается промежуточная сумма.

Результат работы

1. Вопрос: len([1, 2, 3, 4, 5], RES).

Ответ: RES=5

1. Вопрос: len([1], RES).

Ответ: RES=1

1. Вопрос: len([], RES).

Ответ: RES=0

1. Вопрос: sum([2, 4, 1], RES).

Ответ: RES=7

1. Вопрос: sum([2], RES).

Ответ: RES=2

1. Вопрос: sum([], RES).

Ответ: RES=0

1. Вопрос: sum\_odd([1, 5, 2, 4, 3], RES).

Ответ: RES=9

1. Вопрос: sum\_odd([1, 5, 2], RES).

Ответ: RES=5

1. Вопрос: sum\_odd([1], RES).

Ответ: RES=0

1. Вопрос: sum\_odd([], RES).

Ответ: RES=0

Таблица

**1 sum**(INT\_LIST, RES) :- **sum\_inner**(INT\_LIST, RES, **0**).

**2 sum\_inner**([], RES, RES) :- !.

**3 sum\_inner**([Head|Tail], RES, CUR\_SUM) :-

CUR\_SUM1 = CUR\_SUM + Head,

**sum\_inner**(Tail, RES, CUR\_SUM1).

Вопрос: sum([2, 4, 1], RES).

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Текущая резольвента – ТР | ТЦ, выбираемые правила: сравниваемые термы,  подстановка | Дальнейшие действия с комментариями |
| шаг 1 | ТР:  sum([2, 4, 1], RES). | ТЦ: sum([2, 4, 1], RES)  ПР1: [2, 4, 1] = INT\_LIST, RES = RES  Успех (подобрано знание).  Подстановка:  {INT\_LIST = [2, 4, 1],  RES = RES}. | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 2 | ТР:  sum\_inner([2, 4, 1], RES, 0). | ТЦ: sum\_inner([2, 4, 1], RES, 0)  ПР2: [2, 4, 1] = [].  Неудача, унификация невозможна | возврат к ТЦ, метка переносится ниже. |
| ТЦ: sum\_inner([2, 4, 1], RES, 0)  ПР3: [2, 4, 1] = [Head|Tail], RES = RES, 0 = CUR\_SUM. Успех (подобрано знание).  Подстановка:  {Head = 2,  Tail = [4, 1],  RES = RES,  CUR\_SUM = 0} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 3 | ТР:  CUR\_SUM1 = 0 + 2,  sum\_inner([4, 1], RES, CUR\_SUM1) | ТЦ: CUR\_SUM1 = 0 + 2  Подстановка:  {CUR\_SUM1 = 2} | Прямой ход. Применение подстановки к резольвенте. |
| шаг 4 | ТР:  sum\_inner([4, 1], RES, 2) | ТЦ: sum\_inner([4, 1], RES, 2)  ПР2: [4, 1] = [].  Неудача, унификация невозможна | возврат к ТЦ, метка переносится ниже. |
| ТЦ: sum\_inner([4, 1], RES, 2)  ПР3: [4, 1] = [Head|Tail], RES = RES, 2 = CUR\_SUM.  Успех (подобрано знание).  Подстановка:  {Head = 4,  Tail = [1],  RES = RES,  CUR\_SUM = 2} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 5 | ТР:  CUR\_SUM1 = 2 + 4,  sum\_inner([1], RES, CUR\_SUM1) | ТЦ: CUR\_SUM1 = 2 + 4  Подстановка:  {CUR\_SUM1 = 6} | Прямой ход. Применение подстановки к резольвенте. |
| шаг 6 | ТР:  sum\_inner([1], RES, 6). | ТЦ: sum\_inner([1], RES, 6).  ПР2: [1] = [].  Неудача, унификация невозможна | возврат к ТЦ, метка переносится ниже. |
| ТЦ: sum\_inner([1], RES, 6).  ПР3: [1] = [Head|Tail], RES = RES, 2 = CUR\_SUM.  Успех (подобрано знание).  Подстановка:  {Head = 1,  Tail = [],  RES = RES,  CUR\_SUM = 6} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 7 | ТР:  CUR\_SUM1 = 6 + 1,  sum\_inner([], RES, CUR\_SUM1) | ТЦ: CUR\_SUM1 = 6 + 1  Подстановка:  {CUR\_SUM1 = 7} | Прямой ход. Применение подстановки к резольвенте. |
| шаг 8 | ТР:  sum\_inner([], RES, 7) | ТЦ: sum\_inner([], RES, 7)  ПР2: [] = [], RES = RES, 7 = RES  Подстановка:  {[] = [],  RES = RES,  RES = 7} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 9 | ТР:  ! | Встречен символ отсечения. | Отсечение. Вывод результата. Конец. |

Вывод

Рассматривая вариант работы программы в Prolog с поиском всех возможных решений, алгоритм поиска решений осуществляет полный перебор всех знаний. Для достижения большей эффективности стоит использовать предикат отсечения. Он позволяет не рассматривать бесперспективные ветви поиска. Также на эффективность влияет формулировка правил и фактов. Если тело правило содержит проверки его применимости, их стоит внести в заголовок в качестве аргументов.