|  |  |
| --- | --- |
| 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 № 20**

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

Москва.

2020 г.

**Задание**

Используя хвостовую рекурсию, разработать, комментируя аргументы, эффективную программу, позволяющую:

1. Сформировать список из элементов числового списка, больших заданного значения;
2. Сформировать список из элементов, стоящих на нечетных позициях исходного списка (нумерация от 0);
3. Удалить заданный элемент из списка (один или все вхождения);
4. Преобразовать список в множество (можно использовать ранее разработанные процедуры). Убедиться в правильности результатов

Для одного из вариантов ВОПРОСА и 1-ого задания составить таблицу, отражающую конкретный порядок работы систем

## Ответы на вопросы

#### Как организуется хвостовая рекурсия в Prolog?

Хвостовая рекурсия организуется следующим образом: рекурсивный вызов - последний в каждом правиле; нужно избавиться от точек возврата с помощью предиката отсечения (!), чтобы исключить возможные альтернативы.

#### Какое первое состояние резольвенты?

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

1. **Каким способом можно разделить список на части, какие, требования к частям?** Список может быть разделен на «начало» и «конец». Начало списка – группа первых элементов(один и более), конец – обязательно один список. Разделить список можно с помощью вертикальной черты (|)

Пример: [1, 2, 3] -> [1 | [2, 3]] -> [1, 2 | [3]] -> [1, 2, 3 | []].

#### Как выделить за один шаг первые два подряд идущих элемента списка? Как выделить 1-й и 3-й элемент за один шаг?

Два подряд идущий элемента могут быть выделены с помощью вертикальной черты и именованных переменных, например

get\_fisrt\_and\_second([H1, H2 | T]),

Первый и третий с помощью вертикальной черты, именованных переменных и анонимной. get\_fisrt\_and\_third([H1, \_, H3 | T])

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

## domains

list = integer\*

## predicates

/\* 1 арг – список, на обработку, 2 арг. – результирующий список, 3 арг. – число, с которым сравниваются элементы \*/

add(list, list, integer)

/\*1 арг – список, на обработку, 2 арг. – результирующий список\*/

odd(list, list)

/\* 1 арг – список, на обработку, 2 арг. – результирующий список, 3 арг. – число, с которым сравниваются элементы спика\*/

delete\_all(list, list, integer)

## clauses

/\*1 арг – список, на обработку, 2 арг. – результирующий список\*/

set(list, list)

/\*задание 1\*/ add([],[], \_).

add([H|T], [H|T2], El) :- H > El, add(T, T2, El), !.

add([\_|T], T2, EL) :- add(T, T2, El).

/\*задание 2\*/ odd([], []).

odd([\_,H|T],[H|T2]) :- odd(T, T2), !.

odd([\_|T], T2) :- odd(T, T2).

/\*задание 3\*/ delete\_all([], [], \_).

delete\_all([El|T],T2, El) :- delete\_all(T, T2, El), !.

delete\_all([H|T], [H|T2], El):- delete\_all(T, T2, El).

/\*задание 4\*/ set([], []).

set([H|T],[H|T1]) :- delete\_all(T, T2, H), set(T2, T1).

## Примеры целей и результатов работы программы

### 1. **Goal** add([6, 1, 5, 0, -3, 3, 8], Result, 3).

**Result** Result=[6, 5, 8]

2. **Goal** add([1, 2, 3, 4], Result, 5).

**Result** Result=[]

3. **Goal** odd([1, 2, 3, 7, 1], Result).

**Result** Result=[2, 7]

1. **Goal** odd([1], Result).

**Result** Result=[]

### **Goal** odd([1, 2], Result).

**Result** Result=[2]

### **Goal** delete\_all([1, 2, 3, 2, 5], Result, 2).

**Result** Result=[1,3,5]

### 7. **Goal** set([1, 2, 2, 3, 4, 5, 2, 2, 3], Result).

**Result** Result=[1,2,3,4,5]

## Описание порядка поиска объектов Текст процедуры:

**I** add([],[], \_).

1. add([H|T], [H|T2], El) :- H > El, add(T, T2, El), !.
2. add([\_|T], T2, EL) :- add(T, T2, El).

**Goal** add([4], Result, 2).

## Вопрос

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Текущая резольвента – ТР | ТЦ, выбираемые правила: сравниваемые термы,  подстановка | Дальнейшие действия с комментариями |
| Шаг1 | add([4], Result, 2) | ТЦ: add([4], Result, 2). | Поиск знания с начала базы знаний. |
|  | add([4], Result, 2) | ТЦ: add([4], Result, 2).  Сравниваниемые термы: add([4], Result, 2)  ПРI: add([],[], \_).  Результат: унификация невозможна | Возврат к ТЦ, метка переносится ниже. |
|  | add([4], Result, 2) | ТЦ: add([4], Result, 2).  Сравниваниемые термы:  add([4], Result, 2)  ПРII: add([H|T], [H|T2], El)  Результат: успех (подобрано знание) Подстановка: {H=4, T=[], Result=[4|T2], El=2} | Проверка тела ПРII. Изменение резольвенты - замена цели на тело правила. Применение подстановки к резольвенте. |

|  |  |  |  |
| --- | --- | --- | --- |
| Шаг2 | 4 > 2  add([], T2, 2)  !. | ТЦ: 4 > 2 - истинно | Прямой ход. Истинная цель удалена из резольвенты. |
| Шаг3 | add([], T2, 2)  !. | ТЦ: add([], T2, 2) | Поиск знания с начала базы знаний. |
|  | add([], T2, 2)  !. | ТЦ: add([], T2, 2)  Сравниваниемые термы: add([], T2, 2)  ПРI: add([],[], \_).  Результат: успех (подобрано знание) Подстановка: {T2=[]} | Прямой ход. |
| Шаг4 | !. | Выполнение отсечения | Прямой ход. |
|  | Пусто |  | Успех, резольвента пуста –> откат.  Попытка отката ! приводит к завершению работы процедуры (правило доказано).  Завершение работы программы: Result = [4|[]]. |

**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.