|  |  |
| --- | --- |
| 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 № 20*

*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, 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?**

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 может находиться предикат отсечения.

* **Какое первое состояние резольвенты?**

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

* **Каким способом можно разделить список на части, какие, требования к частям?**

Список может быть разделен на начало и конец с помощью вертикальной линии, левая часть(начало), может состоять из одного и более элементов, правая часть(конец, хвост) состоит из одного списка.

Пример: [1, 2, 3 | [2, 3]], 1, 2 , 3 – начало, [2, 3] – хвост.

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

Можно выделять отдельные элементы списка с помощью вертикальной линии и переменных:

1, 2 элементы: get([H1, H2| T]).

1, 3 элементы: g et([H1,\_, H3| T]).

* **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
* **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  % первый аргумент – число,  % с которым будет производиться сравнение элементов списка   % второй аргумент – список, элементы которого будут сравниваться  % третий аргумент – результирующий список  list\_add\_greater(integer, intList, intList)   % первый аргумент – список, элементы которого будут обрабатываться  % второй аргумент – результирущий список  list\_odd(intList, intList)   % первый аргумент – число,  %с которым будет производиться сравнение элементов списка   % второй аргумент – список, элементы которого будут сравниваться  % третий аргумент – результирующий список  delete\_all(integer, intList, intList)    % первый аргумент – список, элементы которого будут обрабатываться  % второй аргумент – результирущий список (список-множество).  list\_set(intList, intList)   clauses   list\_add\_greater(\_, [],[]).  list\_add\_greater(El, [H|T], [H|T2]) :- H > El,   list\_add\_greater(El, T, T2), !.   list\_add\_greater(El, [\_|T], T2) :- list\_add\_greater(El, T, T2).    list\_odd([], []).  list\_odd([\_,H|T],[H|T2]) :- list\_odd(T, T2), !.  list\_odd([\_|T], T2) :- list\_odd(T, T2).    delete\_all(\_, [], []).  delete\_all(El, [H|T],[H|T2]) :- H<>EL,  delete\_all(El, T, T2),   !.  delete\_all(El, [\_|T], T2):- delete\_all(El, T, T2).    list\_set([], []).  list\_set([H|T],[H|T1]) :- delete\_all(H, T, T2),  list\_set(T2, T1). |

**Примеры целей и результатов работы программы**

**Goal** list\_add\_greater(1, [6, 1, 5, 0, -3, 3, 8], Res).

**Result** Res=[6,5,3,8]

**Goal** list\_add\_greater(5, [1, 2, 3, 4], Res).

**Result** Res=[]

**Goal** list\_odd([1, 2, 3, 4, 5], Res).

**Result** Res=[2,4]

**Goal** list\_odd([1], Res)

**Result** Res=[]

**Goal** delete\_all(2, [1, 2, 3, 2, 4], Res).

**Result** Res=[1,3,4]

**Goal** delete\_all([1, 2, 3, 2, 4], List, 6).

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

**Goal** list\_set([1, 2, 2, 3, 4, 5, 2, 2, 3], Res)

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

**Goal** list\_set([1, 2, 3, 4], Res)

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

**Описание порядка поиска объектов**

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

1. list\_add\_greater(\_, [],[]).
2. list\_add\_greater(El, [H|T], [H|T2]) :- H > El,

list\_add\_greater(El, T, T2), !.

1. list\_add\_greater(El, [\_|T], T2) :- list\_add\_greater(El, T, T2).

**Вопрос:** list\_add\_greater (0, [1, 2], Res).

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Текущая резольвента – ТР | ТЦ, выбираемые правила: сравниваемые термы,  подстановка | Дальнейшие действия с комментариями |
| Шаг1 | list\_add\_greater (0, [1, 2], Res). | **ТЦ:** list\_add\_greater (0, [1, 2], Res). | Поиск знания с начала базы знаний. |
|  | list\_add\_greater  (0, [1, 2], Res). | **ТЦ:** list\_add\_greater (0, [1, 2], Res).  **Сравниваниемые термы:**  list\_add\_greater (0, [1, 2], Res).  list\_add\_greater (\_, [],[]). (ПРI)  **Результат:** унификация невозможна | Возврат к ТЦ, метка переносится ниже. |
|  | list\_add\_greater  (0, [1, 2], Res). | **ТЦ:** list\_add\_greater (0, [1, 2], Res).  **Сравниваниемые термы**:  list\_add\_greater (0, [1, 2], Res).  list\_add\_greater(El, [H|T], [H|T2]) (ПР2)  **Результат:** успех (подобрано знание)  **Подстановка:** {El=0, H=1, T=[2], Res=[1|T2]} | Проверка тела ПРII |
| Шаг2 | 1 > 0  list\_add\_greater(0, [2], T2)  !.  Резольвента изменилась в 2 этапа. | **ТЦ:** 1 > 0  **Результат:** успех | Пустое тело заменяет цель в резольвенте |
| Шаг3 | list\_add\_greater(0, [2], T2)  !.  Резольвента изменилась в 2 этапа. | **ТЦ:** list\_add\_greater(0, [2], T2) | Поиск знания с начала базы знаний. |
|  | list\_add\_greater(0, [2], T2)  !. | **ТЦ:** list\_add\_greater(0, [2], T2)  **Сравниваниемые термы:**  list\_add\_greater(0, [2], T2)  list\_add\_greater (\_, [],[]). (ПРI)  **Результат:** унификация невозможна | Возврат к ТЦ, метка переносится ниже. |
|  | list\_add\_greater(0, [2], T2)  !. | **ТЦ:** list\_add\_greater(0, [2], T2)  **Сравниваниемые термы:**  list\_add\_greater(0, [2], T2)  list\_add\_greater(El, [H|T], [H|T2]) (ПР2)  **Результат:** успех (подобрано знание)  **Подстановка:** {El=0, H=2, T=[], T2=[2|T2]} | Проверка тела ПРII |
| Шаг4 | **2 > 0**  **list\_add\_greater(0, [], T2)**  **!**  !  Резольвента изменилась в 2 этапа. | **ТЦ:** 2 > 0  **Результат**: успех | Пустое тело заменяет цель в резольвенте |
| Шаг5 | **list\_add\_greater(0, [], T2)**  **!**  !  Резольвента изменилась в 2 этапа. | **ТЦ**: list\_add\_greater(0, [], T2) | Поиск знания с начала базы знаний. |
| Шаг6 | **list\_add\_greater(0, [], T2)**  **!**  ! | **ТЦ**: list\_add\_greater(0, [], T2)  **Сравниваниемые термы:**  list\_add\_greater(0, [], T2)  list\_add\_greater (\_, [],[]). (ПРI)  **Результат**: успех (подобрано знание)  Подстановка: {T2=[]} | Пустое тело заменяет цель в резольвенте |
| Шаг7 | **!**  ! | **ТЦ**:!  Выполнение отсечения  **Результат**: Успех | Пустое тело заменяет цель в резольвенте. |
| Шаг8 | ! | **ТЦ**:!  Выполнение отсечения  **Результат**: Успех | Правило list\_add\_greater(0, [2], T2) доказано,  ответ: T2=[]  Пустое тело заменяет цель в резольвенте. |
| Шаг9 | Пусто |  | Правило доказано,  ответ: T2=[2|[]]  Попытка отката ! приводит к завершению процедуры  Res=[1|[2|[]]] = [1, 2] |

**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.