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

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

1. **по теме** “ **Формирование и модификация списков на Prolog**”

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

*2020 г.*

**Цель работы** – изучить способы формирования и модификации списков в Prolog, эффективные методы обработки списков и порядок реализации рекурсивных программ.

**Задачи работы**: приобрести навыки формирования и модификации 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. Как организуется хвостовая рекурсия в Prolog?
2. Какое первое состояние резольвенты?
3. Каким способом можно разделить список на части, какие, требования к частям?
4. Как выделить за один шаг первые два подряд идущих элемента списка? Как выделить 1-й и 3-й элемент за один шаг?
5. Как формируется новое состояние резольвенты?
6. Когда останавливается работа системы? Как это определяется на формальном уровне?

**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 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 соответствующий вывод: успех или нет –и почему.

**Текст процедуры …; Вопрос:…..**

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Текущая резольвента – ТР | ТЦ, выбираемые правила: сравниваемые термы,  подстановка | Дальнейшие действия с комментариями |
| шаг1 | … | … | … |
| … | … | … | … |

Вопросы

1. **Как организуется хвостовая рекурсия в Prolog?**

Для организации хвостовой рекурсии в Prolog рекурсивный вызов определяемого предиката должен быть последней подцелью в теле рекурсивного правила.

1. **Какое первое состояние резольвенты?**

Первое состояние резольвенты – это вопрос.

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

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. Хвост должен быть обязательно списком. Голова списка – это группа не менее, чем из одного элемента.

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

Чтобы выделить за один шаг первые два элемента списка: [elem1, elem2 | Tail], где elem1 – это первый элемент списка, elem2 – второй элемент списка, а Tail – оставшийся список. Для того, чтобы получить 1-й и 3-й элемент: [elem1, \_, elem3 | Tail], где elem1 – это первый элемент списка, elem3 – это третий элемент, а Tail – оставшиеся список.

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. **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

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

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

**new\_odd\_list**(int\_list, int\_list).

**new\_odd\_list\_inner**(int\_list, int\_list, int\_list).

**remove\_l**(int\_list, int\_list, num).

**remove\_l\_inner**(int\_list, int\_list, int\_list, num).

**form\_set**(int\_list, int\_list).

**form\_set\_inner**(int\_list, int\_list, int\_list).

**reverse**(int\_list, int\_list, int\_list).

clauses

**new\_list**(INT\_LIST, Filter\_num, RES) :-

**new\_list\_inner**(INT\_LIST, Filter\_num, RES, []).

**new\_list\_inner**([], **\_**, RES, RES) :- !.

**new\_list\_inner**([Head|Tail], Filter\_num, RES, Cur\_list) :-

Head > Filter\_num,

**new\_list\_inner**(Tail, Filter\_num, RES, [Head|Cur\_list]), !.

**new\_list\_inner**([**\_**|Tail], Filter\_num, RES, Cur\_list) :-

**new\_list\_inner**(Tail, Filter\_num, RES, Cur\_list).

**new\_odd\_list**(INT\_LIST, RES) :-

**new\_odd\_list\_inner**(INT\_LIST, RES, []).

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

**new\_odd\_list\_inner**([Num], [NUM|CUR\_LIST], CUR\_LIST) :- !.

**new\_odd\_list\_inner**([Num, **\_**|Tail], RES, CUR\_LIST) :-

**new\_odd\_list\_inner**(Tail, RES, [Num|CUR\_LIST]).

**remove\_l**(INT\_LIST, RES, NUM) :-

**remove\_l\_inner**(INT\_LIST, RES, [], NUM).

**remove\_l\_inner**([], RES, RES, **\_**) :- !.

**remove\_l\_inner**([Head|Tail], RES, Cur\_list, NUM) :-

Head <> NUM,

**remove\_l\_inner**(Tail, RES, [Head|Cur\_list], NUM),!.

**remove\_l\_inner**([**\_**|Tail], RES, Cur\_list, NUM) :-

**remove\_l\_inner**(Tail, RES, Cur\_list, NUM).

**form\_set**(INT\_LIST, RES) :- **form\_set\_inner**(INT\_LIST, RES, []).

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

**form\_set\_inner**([Head|Tail], RES, Cur\_list) :-

**remove\_l\_inner**(Tail, Cur\_list1, [], Head),

**form\_set\_inner**(Cur\_list1, RES, [Head|Cur\_list]).

**reverse**([], RES, RES) :- !.

**reverse**([Head|Tail], RES, CUR\_LIST) :-

**reverse**(Tail, RES, [Head|CUR\_LIST]).

goal

**new\_list**([**1**, **2**, **3**, **4**, **5**, **2**], **2**, RES).

% new\_odd\_list([1, 2, 3, 4, 5], RES).

% remove\_l([1, 2, 3, 4, 5, 2], RES, 2).

% form\_set([1, 2], RES).

Обозначения

1. new\_list –правило-обёртка, которое получает 3 аргумента:

* исходный список;
* число, с которым будет сравниваться элементы списка;
* список, в который будет записан результат.

1. new\_list\_inner – привило, в котором осуществляется хвостовая рекурсия и которое получает 4 аргумента:

* список, по которому производится перебор;
* число, с которым будет сравниваться элементы списка;
* список, в который будет записан результат;
* список, с промежуточным результатом.

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

* исходный список;
* список, в который будет записан результат.

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

* список, по которому производится перебор;
* список, в который будет записан результат.
* список, с промежуточным результатом.

1. remove\_l –правило-обёртка, которое получает 3 аргумента:

* исходный список;
* список, в который будет записан результат;
* элемент, который нужно удалить из списка.

1. remove\_l\_inner – привило, в котором осуществляется хвостовая рекурсия и которое получает 4 аргумента:

* список, по которому производится перебор;
* список, в который будет записан результат;
* список, с промежуточным результатом,
* элемент, который нужно удалить из списка.

1. form\_set –правило-обёртка, которое получает 3 аргумента:

* исходный список;
* список, в который будет записан результат;

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

* список, по которому производится перебор;
* список, в который будет записан результат;
* список, с промежуточным результатом.

Результат работы

1. Вопрос: new\_list([1, 2, 3, 4, 5, 2], 2, RES).

Ответ: RES=[5,4,3]

1. Вопрос: new\_list([1, 1, 0], 3, RES).

Ответ: RES=[]

1. Вопрос: new\_list([], 3, RES).

Ответ: RES=[]

1. Вопрос: new\_list([1, 2, 3, 4, 2], 2, RES).

Ответ: RES=[4, 3]

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

Ответ: RES=[5,3,1]

1. Вопрос: new\_odd\_list([1], RES).

Ответ: RES=[1]

1. Вопрос: new\_odd\_list([], RES).

Ответ: RES=[]

1. Вопрос: remove\_l([1, 2, 3, 4, 5, 2], RES, 2).

Ответ: RES=[5,4,3,1]

1. Вопрос: remove\_l([1, 2, 3, 4, 5, 2], RES, 6).

Ответ: RES=[2,5,4,3,2,1]

1. Вопрос: form\_set([1, 2, 2, 5], RES).

Ответ: RES=[2,5,1]

1. Вопрос: form\_set([1, 2, 5], RES).

Ответ: RES=[2,5,1]

1. Вопрос: form\_set([1, 2, 5, 2, 1, 5], RES).

Ответ: RES=[2,5,1]

Таблица

**1 new\_list**(INT\_LIST, Filter\_num, RES) :-

**new\_list\_inner**(INT\_LIST, Filter\_num, RES, []).

**2 new\_list\_inner**([], **\_**, RES, RES) :- !.

**3** **new\_list\_inner**([Head|Tail], Filter\_num, RES, Cur\_list) :-

Head > Filter\_num,

**new\_list\_inner**(Tail, Filter\_num, RES, [Head|Cur\_list]), !.

**4 new\_list\_inner**([**\_**|Tail], Filter\_num, RES, Cur\_list) :-

**new\_list\_inner**(Tail, Filter\_num, RES, Cur\_list).

Вопрос: new\_list([1, 2, 3, 4, 2], 2, RES).

|  |  |  |  |
| --- | --- | --- | --- |
| № шага | Текущая резольвента – ТР | ТЦ, выбираемые правила: сравниваемые термы,  подстановка | Дальнейшие действия с комментариями |
| шаг 1 | ТР:  new\_list([1, 2, 3, 4, 2], 2, RES). | ТЦ: new\_list([1, 2, 3, 4, 2], 2, RES).  ПР1: [1, 2, 3, 4, 2] = INT\_LIST, 2 = Filter\_num, RES = RES.  Подстановка:  {INT\_LIST = [1, 2, 3, 4, 2], Filter\_num = 2, RES = RES}. | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 2 | ТР:  new\_list\_inner([1, 2, 3, 4, 2], 2, RES, []). | ТЦ: new\_list\_inner([1, 2, 3, 4, 2], 2, RES, []).  ПР2: [2, 4, 1] = [].  Неудача, унификация невозможна | возврат к ТЦ, метка переносится ниже. |
| ТЦ: new\_list\_inner([1, 2, 3, 4, 2], 2, RES, []).  ПР3: [1, 2, 3, 4, 2] = [Head|Tail], Filter\_num = 2, RES = RES, [] = Cur\_list.  Успех (подобрано знание).  Подстановка:  {Head = 1,  Tail = [2, 3, 4, 2],  Filter\_num = 2,  RES = RES,  Cur\_list = []} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 3 | ТР:  1 > 2,  new\_list\_inner([2, 3, 4, 2], 2, RES, [1|[]]), !. | ТЦ: 1 > 2,  Сравнение 1 и 2 на >(1, 2).  Результат: “нет” | Откат к предыдущему состоянию. Переход к следующему предложению. |
| шаг 5 | ТР:  new\_list\_inner([1, 2, 3, 4, 2], 2, RES, []). | ТЦ: new\_list\_inner([1, 2, 3, 4, 2], 2, RES, []).  ПР4: [1, 2, 3, 4, 2] = [\_|Tail], 2 = Filter\_num, RES, [].  Успех (подобрано знание).  Подстановка: {\_ = 1,  Tail = [2, 3, 4, 2],  Filter\_num = 2  RES = RES,  Cur\_list = []} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 6 | ТР:  new\_list\_inner([2, 3, 4, 2], 2, RES, []). | ТЦ: new\_list\_inner([2, 3, 4, 2], 2, RES, []).  ПР2: [2, 3, 4, 2] = []  Неудача, унификация невозможна | возврат к ТЦ, метка переносится ниже. |
| ТЦ: new\_list\_inner([2, 3, 4, 2], 2, RES, []).  ПР3: [2, 3, 4, 2] = [Head|Tail], 2 = Filter\_num, RES, [].  Успех (подобрано знание).  Подстановка: {Head = 2,  Tail = [3, 4, 2],  Filter\_num = 2,  RES = RES,  Cur\_list = []} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 7 | ТР:  2 > 2,  new\_list\_inner([3, 4, 2], 2, RES, [2|[]]), !. | ТЦ: 2 > 2,  Сравнение 2 и 2 на >(2, 2).  Результат: “нет” | Откат к предыдущему состоянию. Переход к следующему предложению. |
| шаг 8 | ТР:  new\_list\_inner([2, 3, 4, 2], 2, RES, []). | ТЦ: new\_list\_inner([2, 3, 4, 2], 2, RES, []).  ПР4: [2, 3, 4, 2] = [\_|Tail], 2 = Filter\_num, RES, [].  Успех (подобрано знание).  Подстановка: {\_ = 2,  Tail = [3, 4, 2],  Filter\_num = 2,  RES = RES,  Cur\_list = []} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 9 | ТР:  new\_list\_inner([3, 4, 2], 2, RES, []). | ТЦ: new\_list\_inner([3, 4, 2], 2, RES, []).  ПР2: [3, 4, 2] = []  Неудача, унификация невозможна | возврат к ТЦ, метка переносится ниже. |
| ТЦ: new\_list\_inner([3, 4, 2], 2, RES, []).  ПР3: [3, 4, 2] = [Head|Tail], 2 = Filter\_num, RES, [].  Успех (подобрано знание).  Подстановка: {Head = 3,  Tail = [4, 2],  RES = RES,  Cur\_list = []} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 10 | ТР:  3 > 2,  new\_list\_inner([4, 2], 2, RES, [3|[]]), !. | ТЦ: 3 > 2,  Сравнение 3 и 2 на >(3, 2).  Результат: “да” | Прямой ход. Производится редукция. |
| шаг 11 | ТР:  new\_list\_inner([4, 2], 2, RES, [3]). | ТЦ: new\_list\_inner([4, 2], 2, RES, [3]).  ПР2: [3, 4, 2] = []  Неудача, унификация невозможна | возврат к ТЦ, метка переносится ниже. |
| ТЦ: new\_list\_inner([3, 4, 2], 2, RES, []).  ПР3: [4, 2] = [Head|Tail], 2 = Filter\_num, RES, [].  Успех (подобрано знание).  Подстановка: {Head = 4,  Tail = [2],  Filter\_num = 2,  RES = RES,  Cur\_list = [3]} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 12 | ТР:  4 > 2,  new\_list\_inner([2], 2, RES, [4|[3]]), !. | ТЦ: 4 > 2,  Сравнение 4 и 2 на >(4, 2).  Результат: “да” | Прямой ход. Производится редукция. |
| шаг 13 | ТР:  new\_list\_inner([2], 2, RES, [4, 3]). | ТЦ: new\_list\_inner([2], 2, RES, [3]).  ПР2: [2] = []  Неудача, унификация невозможна | возврат к ТЦ, метка переносится ниже. |
| ТЦ: new\_list\_inner([2], 2, RES, []).  ПР3: [2] = [Head|Tail], 2 = Filter\_num, RES, [].  Успех (подобрано знание).  Подстановка: {Head = 2,  Tail = [],  Filter\_num = 2,  RES = RES,  Cur\_list = [4, 3]} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 14 | ТР:  2 > 2,  new\_list\_inner([], 2, RES, [2|[4, 3]]), !. | ТЦ: 2 > 2,  Сравнение 2 и 2 на >(2, 2).  Результат: “нет” | Откат к предыдущему состоянию. Переход к следующему предложению. |
| шаг 15 | ТР:  new\_list\_inner([], 2, RES, [4, 3]). | ТЦ: new\_list\_inner([2], 2, RES, [4, 3]).  ПР4: [2] = [\_|Tail], 2 = Filter\_num, RES, [4, 3].  Успех (подобрано знание).  Подстановка: {\_ = 2,  Tail = [],  Filter\_num = 2,  RES = RES,  Cur\_list = [4, 3]} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 16 | ТР:  new\_list\_inner([], 2, RES, [4, 3]). | ТЦ: new\_list\_inner([], 2, RES, [4, 3]).  ПР2: [] = []  Успех (подобрано знание).  Подстановка: {[] = [],  \_ = 2,  RES = RES,  RES = [4, 3]} | Прямой ход. Замена цели на тело подобранного правила. Применение подстановки к резольвенте. |
| шаг 18 | 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.