МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Логическое программирование»
Тема: Рекурсия и Структуры данных
Вариант 3.

Студент гр. 0303 _____ Бодунов П.А.
Студент гр. 0303 _____ Болкунов В.О.
Студент гр. 0303 _____ Калмак Д.А.
Преподаватель _____ Родионов С.В.

Санкт-Петербург
2024

Цель работы.

Целью работы является изучение особенностей реализации рекурсии на языке Пролог, освоение принципов решения типовых логических программ.

Задачи.

Для достижения поставленной цели требуется решить следующие задачи:

1) Изучить теоретический материал.

2) Создать правила в соответствии с вариантом задания и общей формулировкой задачи (п.3).

3) Проверить выполнение программы.

4) Составить отчет о выполнении работы.

5) Представить на проверку файл отчета и файл текста программы на языке GNU Prolog, решающей поставленные задачи.

Номер варианта и текст варианта задания должны быть представлены в форме комментариев в тексте программы. Номер группы и номер варианта должны присутствовать в имени файла с текстом программы.

Основные теоретические положения.

Рассмотрим несколько вариантов использования рекурсивного вызова на языке Пролог применительно к спискам.

Принадлежность списку. Сформулируем задачу проверки принадлежности данного терма списку.

Граничное условие:

Терм R содержится в списке [H|T], если R=H.

Рекурсивное условие:

Терм R содержится в списке [H|T], если R содержится в списке T.

Первый вариант записи определения на Прологе имеет вид:

содержится (R, L) :- L=[H | T], H=R.

содержится (R, L) :- L=[H | T], содержится (R, T).

Цель L=[H | T] в теле обоих утверждений служит для того, чтобы разделить список L на голову и хвост.

Можно улучшить программу, если учесть тот факт, что Пролог сначала сопоставляет с целью голову утверждения, а затем пытается согласовать его тело. Новая процедура, которую мы назовем "принадлежит", определяется таким образом:

принадлежит (R, [R | T]).

принадлежит (R, [H | T]) :- принадлежит (R, T).

На запрос

?- принадлежит(a, [a, Ь, c]).

будет получен ответ

да

на запрос

?- принадлежит(b, [a, b, c]).

- ответ

да

но на запрос

?- принадлежит(d, (a, b, c)).

Пролог дает ответ

нет

В большинстве реализации Пролога предикат «принадлежит» является встроенным.

Соединение двух списков. Задача присоединения списка Q к списку P, в результате чего получается список R, формулируется следующим образом:

Граничное условие:

Присоединение списка Q к [] дает Q.

Рекурсивное условие:

Присоединение списка Q к концу списка P выполняется так: Q присоединяется к хвосту P, а затем спереди добавляется голова P.

Определение можно непосредственно написать на Прологе:

соединить([],Q,Q).

соединить(P,Q,R) :- P=[HP | TP], соединить(TP, Q, TR), R=[HP | TR].

Однако, как и в предыдущем примере, воспользуемся тем, что Пролог сопоставляет с целью голову утверждения, прежде чем пытаться согласовать тело:

присоединить([] ,Q,Q).

присоединить(HP | TP], Q, [HP | TR]) :- присоединить (TP, Q, TR).

На запрос

?- присоединить [a, b, c], [d, e], L).

будет получен ответ

L = [a, b, c, d].

но на запрос

?- присоединить([a, b], [c, d], [e, f]).

ответом будет No

Часто процедура «присоединить» используется для получения списков, находящихся слева и справа от данного элемента:

присоединить (L [джим, р], [джек,.билл, джим, тим, джим, боб]) .

L = [джек, билл]

R = [тим, джим, боб]

другие решения (да/нет)? да

L=[джек, билл, джим, тим]

R=[боб]

другие решения (да/нет)? да

других решений нет

Индексирование списка. Задача получения N-го терма в списке определяется следующим образом:

Граничное условие:

Первый терм в списке [H | T] есть H.

Рекурсивное условие:

N-й терм в списке [H | T] является (N-1)-м термом в списке T.

Данному определению соответствует программа:

/* Граничное условие:

получить ([H | T], 1, H).

/* Рекурсивное условие:

получить([H | T], N, Y) :- M is N - 1, получить (T, M ,Y).

Задание.

Задание 1, Списки

Проверить, является ли заданный список "палиндромным" (симметричным)

```
?- palind_list([1,2,3,4,5,4,3,2,1])
Yes
```

Задание 2, Деревья

Напишите предикат, проверяющий, является ли заданное бинарное дерево двоичным справочником (в каждом узле - в левом поддереве - все элементы, меньшие узлового, в правом - большие либо равные узловому)

```
?- isBinaryDict(
     tr(  5,
          tr(4, nil, nil),
          tr(  8,
               tr(  6,
                    tr(3, nil, nil),
                    nil
               ),
               tr(9, nil, nil)
          )
     )
).
No.
```

Выполнение работы.

1. Порядок выполнения

Задание 1, Списки

Были созданы правила: palind_list, rev, rev_rec.

Правило *palind_list(X)* проверяет, является ли слово палиндромом, то есть читается с обеих сторон одинаково. Для этого создано правило *rev(L, R)*, которое запускает рекурсивное правило *rev_rec(L, R, [])*., где L - исходное слово, R - результат с исходным словом, написанным в обратном порядке (на начале неопределенный), а пустой список является временный списком для обратного написания исходного слова. В правиле *palind_list(X) :- rev(X, X).* в *rev* два раза передаётся список X, что позволяет сразу сравнить список с его перевёрнутой версией.

Создано правило *rev_rec([X | Xs], R, T) :- rev_rec(Xs, R, [X | T])*. в котором рекурсивно извлекается голова исходного слова и хвост. Голова переносится в временный список T, а хвост переходит на следующую итерацию рекурсии. Рекурсия выполняется до факта *rev_rec([], R, R)*. когда все буквы слова уже были перенесены во временный список, и список с исходным словом стал пустым, тогда R принимает значение со словом, записанным в обратном порядке.

Задание 2, Деревья

Были созданы правила: isBinaryDict, isBinaryDict_rec.

Создано правило *isBinaryDict(T)*, проверяющее является ли бинарное дерево T двоичным справочником, то есть в каждом узле - в левом поддереве - все элементы, меньшие узлового, а в правом - большие либо равные узловому. Для этого создано рекурсивное правило *isBinaryDict_rec(T, Max_num, Min_num)*, где T - бинарное дерево, а второй и третий аргументы предназначены для числа, меньше которого должен

быть узловой элемент, и для числа, больше или равно которого должен быть элемент, соответственно.

Создано правило *isBinaryDict_rec(tr(A, L, R), Max_num, Min_num) :-*

(Max_num = nil, !; A < Max_num),

(Min_num = nil, !; A >= Min_num),

isBinaryDict_rec(L, A, Min_num),

isBinaryDict_rec(R, Max_num, A).

в котором дерево разделяется на узел A, левое поддерево L и правое поддерево R, Max_num – число, меньше которого должен быть узел, Min_num – число, больше или равен которому должен быть узел.

Первое условие проверяет равенство Max_num и nil, если Max_num не nil тогда проверяется, что *A < Max_num*, иначе если Max_num является nil, то *A < Max_num* не проверяется. Первое условие истинно, если Max_num равен nil или неравенство верно, тогда программа переходит ко второму условию. Второе условие проверяет равенство Min_num и nil, если Min_num не nil, тогда проверяется, что *A >= Min_num*, иначе если Min_num является nil, то *A >= Min_num* не проверяется. Второе условие истинно, если Min_num равен nil или неравенство верно, тогда программа на текущей итерации успешно проверила, что бинарное дерево соблюдает условия для двоичного справочника, и запускается рекурсия сначала в левом поддереве *isBinaryDict_rec(L, A, Min_num)*, при этом передаются значения узла с текущей итерации, как число, меньше которого должны быть узлы, и Max_num занимает место числа, больше или равен которому должен быть узел на следующей итерации. Затем при успешной проверки в левом поддереве запускается рекурсия в правом поддереве *isBinaryDict_rec(R, Max_num, A),* где Max_num становится на место числа, меньше которого должен быть элемент на следующей итерации, а A

занимает место числа, больше или равен которому должен быть узел на следующей итерации.

При выполнении факта *isBinaryDict_rec(nil, _, _) :- !.* программа останавливается, когда узел является неопределенным.

Тестирование программы представлено в разделе 3 Примеры вызова соответствующих правил и результаты выполнения.

2. Текст программы с комментариями

```
/*
Группа 0303. Вариант 3.
Задание 1. Проверить, является ли заданный список "палиндромным" (симметричным).
Реализация инверсии:
Из списка в первом аргументе копируются элементы в обратном порядке
в список в третьем аргументе, после чего полученный список связывается с переданной вторым аргументом переменной.
Если значение второго аргумента равно исходному списку, тогда это палиндром.
*/
rev_rec([], R, R).
rev_rec([X | Xs], R, T) :- rev_rec(Xs, R, [X | T]).

rev(L, R) :- rev_rec(L, R, []).

palind_list(X) :- rev(X, X).

%---

/*
Задание 2. Написать предикат, проверяющий, является ли заданное бинарное дерево двоичным справочником.
Внутренняя реализация для проверки дерева:
Первый аргумент - дерево на текущей итерации разделенное на три аргумента: A - текущий узел дерева, а L и R - левое и правое поддеревья
Второй аргумент - число, меньше которого должен быть элемент
Третий аргумент - число, больше или равен которому должен быть элемент
Программа выводит yes в случае, когда заданное бинарное дерево является двоичным справочником.
*/
isBinaryDict_rec(nil, _, _) :- !.
isBinaryDict_rec(tr(A, L, R), Max_num, Min_num) :-
   (Max_num = nil, !; A < Max_num),
   (Min_num = nil, !; A >= Min_num),
   isBinaryDict_rec(L, A, Min_num),
   isBinaryDict_rec(R, Max_num, A).

isBinaryDict(T) :- isBinaryDict_rec(T, nil, nil).
```

3. Примеры вызова соответствующих правил и результаты выполнения

Вызов правила *palind_list* для проверки списка на палиндром со списком, являющимся палиндромом представлен на рис. 1:

```
| ?- palind_list([1,2,3,4,5,4,3,2,1]).
```

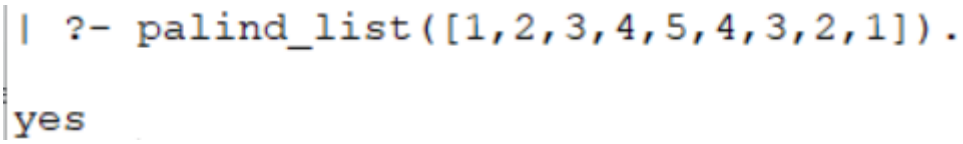

Рисунок 1 - Вызов правила palind_list со списком, являющимся палиндромом

Вызов правила *palind_list* для проверки списка на палиндром со списком, не являющимся палиндромом представлен на рис. 2:

```
| ?- palind_list([1,2,3,4,5,4,3,2,5]).
```

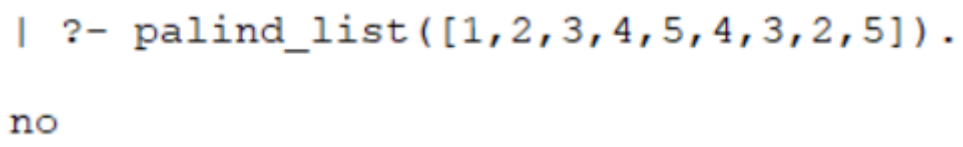

Рисунок 2 - Вызов правила palind_list со списком, не являющимся палиндромом

Вызов правила *isBinaryDict* для определения является ли бинарное дерево двоичным справочником, когда бинарное дерево является двоичным справочником представлен на рис. 3:

```
| ?- isBinaryDict(
    tr(
        10,
        tr(
            5,
            tr(3, nil, nil),
            tr(8, nil, nil)
        ),
        tr(
            15,
            tr(12, nil, nil),
            tr(17, nil, nil)
```

```
        )
    )
).
```

```
| ?- isBinaryDict(
    tr(
        10,
        tr(
            5,
            tr(3, nil, nil),
            tr(8, nil, nil)
        ),
        tr(
            15,
            tr(12, nil, nil),
            tr(17, nil, nil)
        )
    )
).

yes
```

Рисунок 3 - Вызов правила isBinaryDict с бинарним деревом, являющимся двоичным справочником

Вызов правила *isBinaryDict* для бинарного дерева, которое является двоичным справочником при условии равенства элемента в правом поддереве корню представлен на рис. 4:

```
| ?- isBinaryDict(
    tr(
        10,
        tr(
            5,
            tr(3, nil, nil),
            tr(8, nil, nil)
        ),
        tr(
            10,
            nil,
            nil
```

```
            )
        )
    ).
```

```
| ?- isBinaryDict(
    tr(
        10,
        tr(
            5,
            tr(3, nil, nil),
            tr(8, nil, nil)
        ),
        tr(
            10,
            nil,
            nil
        )
    )
).

yes
```

Рисунок 4 - Вызов правила isBinaryDict с бинарним деревом, являющимся двоичным справочником при условии равенства элемента в правом поддереве

Вызов правила *isBinaryDict* для бинарного дерева, которое не является двоичным справочником представлен на рис. 5:

```
| ?- isBinaryDict(tr(5, tr(4, nil, nil), tr(8, tr(6, tr(3, nil, nil), nil), tr(9, nil, nil)))).
```

```
| ?- isBinaryDict(tr(5, tr(4, nil, nil), tr(8, tr(6, tr(3, nil, nil), nil), tr(9, nil, nil)))).
no
```

Рисунок 5 - Вызов правила isBinaryDict с бинарним деревом, не являющимся двоичным справочником

Вызов правила isBinaryDict для предыдущего дерева, но с исправленным узлом представлен на рис. 6:

```
| ?- isBinaryDict(tr(5, tr(4, nil, nil), tr(8, tr(6, tr(5, nil, nil), nil), tr(9, nil, nil)))).
```

```
| ?- isBinaryDict(tr(5, tr(4, nil, nil), tr(8, tr(6, tr(5, nil, nil), nil), tr(9, nil, nil)))).
yes
```

Рисунок 6 - Вызов правила isBinaryDict с бинарним деревом, являющимся двоичным справочником, из предыдущего теста с исправленным элементом

Выводы

В результате выполнения лабораторной работы были описаны правила на языке GNU Prolog, позволяющие решать две задачи: проверку списка на палиндром и проверку, является ли бинарное дерево двоичным справочником. Были приведены примеры вызова правила для проверки палиндрома в случаях, когда список являлся и не являлся им. Так же были приведены примеры вызова правила для проверки, что бинарное дерево является двоичным справочником, в случаях, когда оно являлось и не являлось, при условии равенства элемента в правом поддереве корню.

Роли членов бригады:

Бодунов Пётр 0303 написание кода, написание комментариев в код, оформление отчета.

Болкунов Владислав 0303 написание кода, написание комментариев в код, оформление отчета.

Калмак Даниил 0303 написание кода, написание комментариев в код, оформление отчета.