Лабораторная работа №3.
Определение отношений на основе рекурсивных правил.

Время: 180 мин.

Часть 1. Рекурсия как способ описания мира.

У попа была собака, он её любил,
Она съела кусок мяса, он её убил,
И под камнем схоронил.
А на камне написал:
У попа была собака, …

Описать мир можно задав отношения на основе фактов и/или правил. Например, фактами можно описать всех членов одной семьи или перечислить множество четных чисел в диапазоне от 0 до 99. Правилами можно пользоваться более универсально, отрешившись от конкретных значений аргументов и определив общую для всех зависимость. Например, любой человек в возрасте до 10 лет дитя или все числа, которые нацело делятся на 2, есть четные числа. Но в Прологе возможен ещё один замечательный подход к описанию мира через задание рекурсивных правил, то есть таких которые ссылаются в той или иной форме сами на себя. Например, предком можно считать как непосредственного родителя, так и родителя предка. Такой подход расширяет возможности универсального описания сходных по каким-либо свойствам подмножеств. Например, подмножество предков одного человека, которые являются последовательными предками друг друга или подмножество путей в лабиринте. В общем случае применение рекурсии оправдано при решении таких задач, для которых свойственно то, что решение задачи в целом сводится к решению подобной же задачи, но меньшей размерности.

Для начала очень простой пример, только для того чтобы обозначить основные черты рекурсии. Пусть задан лабиринт из девяти клеток (см. рис). По светлым клеткам перемещаться можно, по иным нет. Необходимо определить существует ли путь от входа (точка «1») до выхода (точка «9»).

Для решения этой задачи опишем:

– совокупность возможных шагов по лабиринту как совокупность пар соседних свободных клеток: `(1,4), (4,5), (5,6), (4,7), step(6,9);`

– процедуру путь, состоящую из двух предложений:

1) путь из точки X в точку Y есть шаг из точки X в точку Y;
2) путь из точки X в точку Y есть шаг из точки X в точку Z и путь из точки Z в точку Y.

Тогда программа будет выглядеть следующим образом:

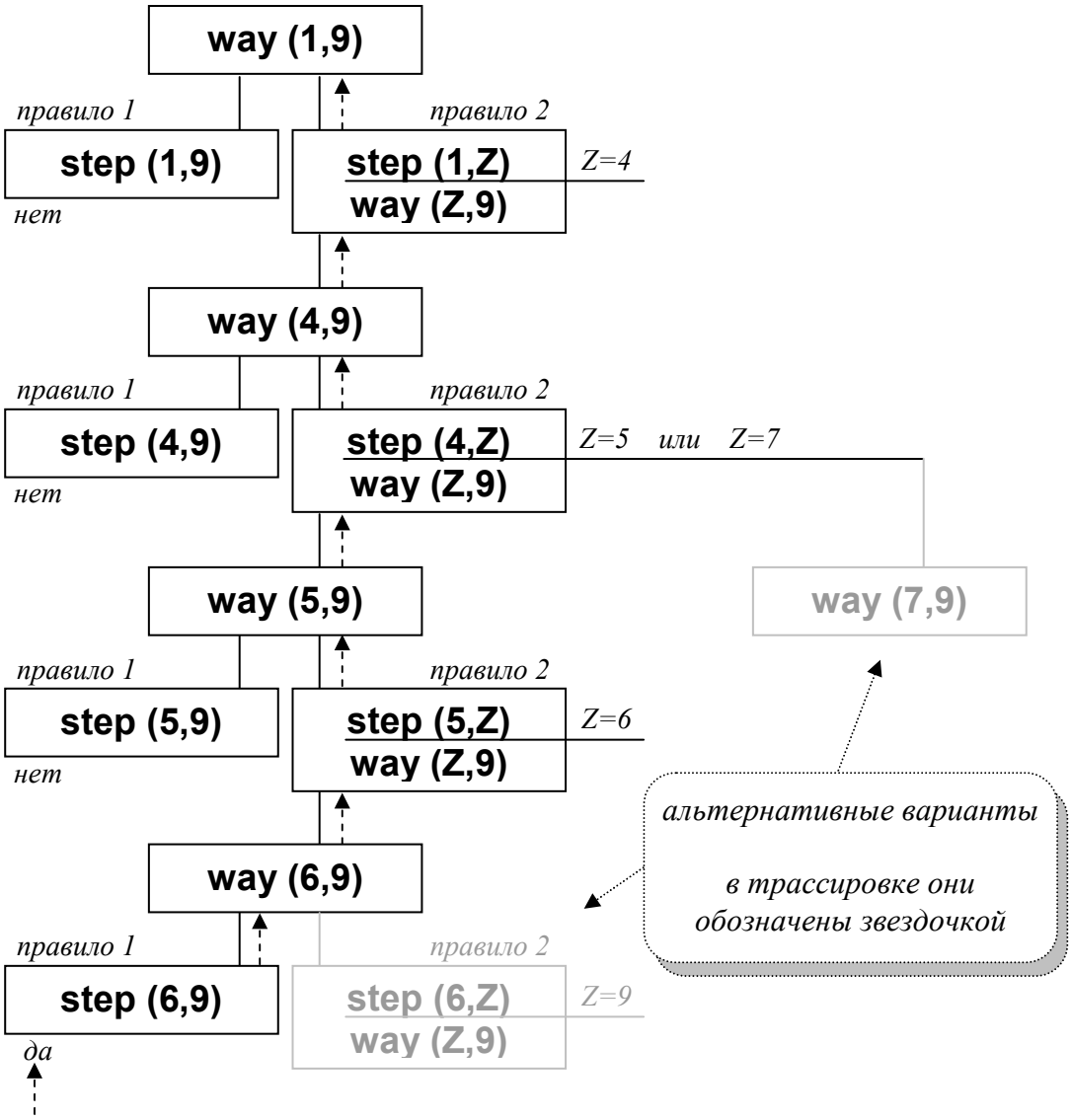
```
PREDICATES
 step(integer,integer)
 way(integer,integer)
CLAUSES
 % факты
 step(1,4). step(4,5).
 step(5,6). step(4,7).
 step(6,9).

 way(X,Y):-step(X,Y).
 way(X,Y):-
      step(X,Z),
      way(Z,Y).
```

1	2	3
4	5	6
7	8	9

Проведем пошаговый разбор процесса унификации цели этой программой и выясним:

– зачем необходимо первое правило в процедуре `way`;

– как пользоваться трассировкой, чтобы понять процесс выполнения программы.



Для лучшего понимания я потратил время на то, чтобы разработать схему трассировки программы (см. выше) и привел саму трассировку (см. скриншот ниже), на которой метками обозначены размышления описанные далее.

Разберитесь со схемой трассировки, так как само по себе грамотное составление трассировки подразумевает, что её составитель разбирается в логическом программировании и, поэтому такое творчество периодически будет являться для вас контрольным мероприятием.

Рекомендации составителя

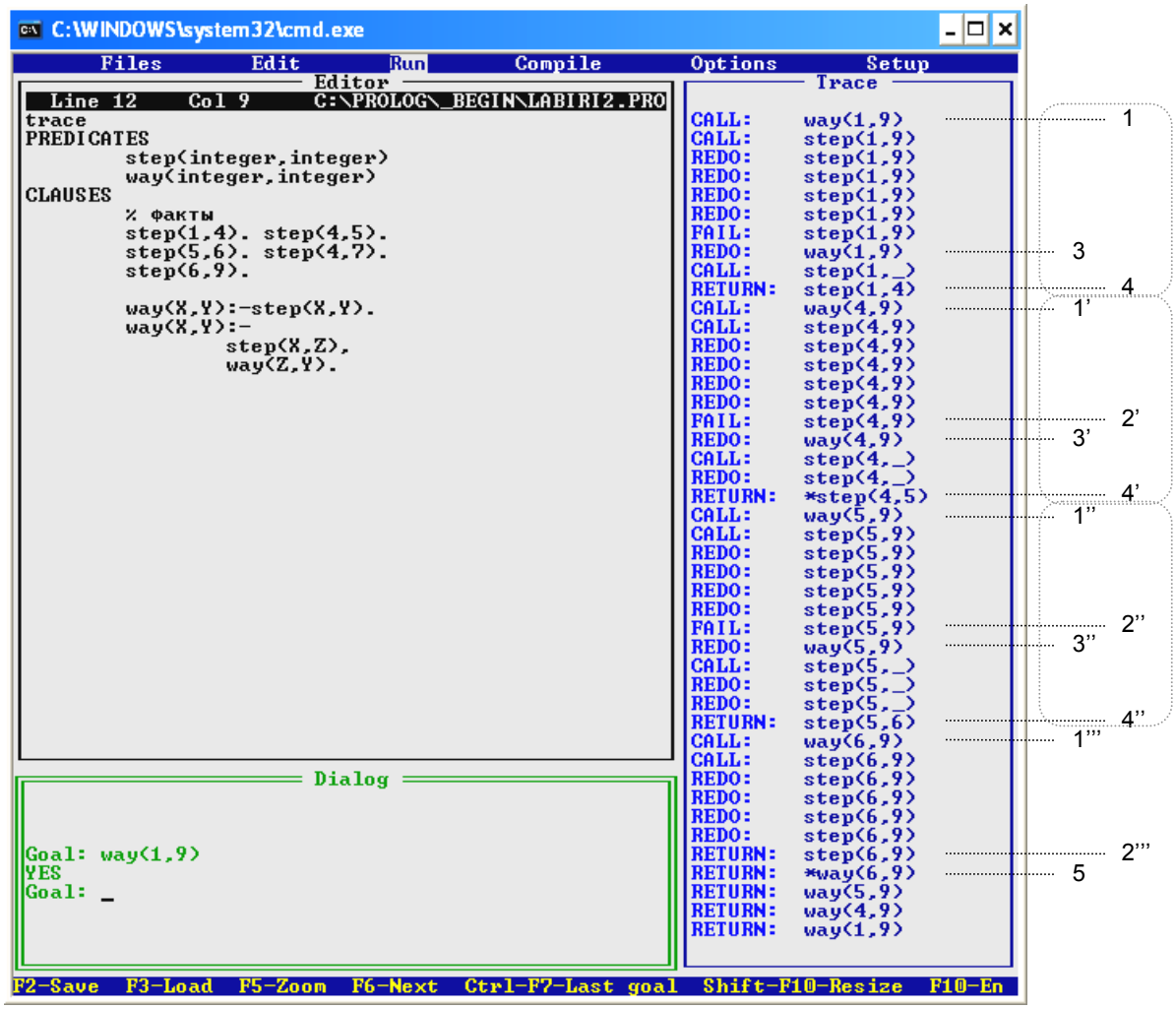
Итак, получив цель way(1,9) Пролог пытается её унифицировать (см. метку 1) последовательно просматривая факты и правила базы знаний. Среди фактов такого не обнаруживается, а вот среди правил оказывается есть кое-что и даже более – есть два правила, в заголовке которых стоит именно предикат way.

Подстановка цели в первое правило вынуждает Пролог проверять его выполнимость. Что состоится, если найдется факт step(1,9). Последовательно сопоставляя его со

всеми фактами Пролог убеждается, что первое правило с указанными аргументами невыполнимо (метка 2).

Тогда Пролог движется далее по программе с надеждой ещё куда-нибудь пристроить первоначальную цель way(1,9). И сталкивается с правилом №2 (метка 3). Оно гласит: путь (1,9) существует, если есть шаг (1,Y) и путь (Y,9). То есть это правило выполнимо, если выполнимы последовательно обе подцели из тела предложения. Приходится их последовательно унифицировать. Унификация первой подцели step(1,Z) заканчивается при анализе первого же факта step(1,4) и Z означивается значением «4» (метка 4). После чего происходит вызов второй подцели way(4,9).

Очевидно, что вторая подцель аналогична исходной цели за исключением значения первого аргумента. Поэтому процесс её унификации аналогичен описанному ранее. И именно поэтому последующие четыре метки в трассировке я обозначил теми же номерами с добавлением штриха (1’, 2’, 3’, 4’). Одно изменение вы можете заметить для метки 4’, а именно, присутствие звездочки в трассировке. Звездочка указывает, что у подцели есть еще альтернативы, к которым возможен возврат. В данном случае у step(4,5) есть альтернативный вариант step(4,7). И если по первому пути унификации основной цели программы успеха не будет достигнуто, тогда Пролог действительно вернется в эту «точку возврата» и продолжит поиски.



Далее аналогично происходит обработка подцели way(5,9) (метки 1’’ … 4’’), что как обычно заканчивается вызовом подцели way(6,9). Но далее уже встречаются отличия, так как правило №1 (*way(X,Y):-step(X,Y).*) с аргументами X=6 и Y=9 в этот раз оказывается выполнимым (метка 2’’’). Пролог последовательно, но в обратном порядке собирает цепочку доказательств подцелей, отметив лишь, что для way(6,9) существует альтернатива в

виде проверки второго правила (метка 5). И в конце концов объявляет Yes, что означает существование пути от точки 1 до точки 9.

Попробуйте самостоятельно запустить программу на доказательство цели way(4,S) и объясните результаты трассировки.

Часть 2. Задания для самостоятельного исполнения.

Задание №1.

Создайте предикат, вычисляющий по натуральному числу N сумму чисел от 1 до N.

Задание №2.

Создайте предикат, вычисляющий неотрицательную степень целого числа.

Задание №3. *(слегка вымышленное)*

Нейтрон, попадая в ядро атома урана поглощается, но вызывает его деление с освобождением 3-х нейтронов, каждый из которых в свою очередь тоже поглощается и инициирует реакцию деления с освобождением 3-х нейтронов. Если от момента освобождения нейтрона до момента освобождения с его помощью трех других проходит 1/100 сек, то сколько нейтронов будет произведено через секунду после освобождения первого нейтрона. Создайте программу расчета N-го члена геометрической прогрессии.

Задание №4.

Двоичное кодирование десятичного числа предполагает, что самый младший разряд имеет вес 1, а каждый последующий старший разряд имеет в два раза больший вес, чем предыдущий. То есть веса разрядов образуют геометрическую прогрессию со знаменателем 2. Таким образом, значение числа определяется суммой весов разрядов означенных единицами. Например, $1001_2=9_{10}$. Определите, какое максимальное натуральное десятичное число можно закодировать N разрядами. Создайте программу расчета суммы N членов геометрической прогрессии.

Задание №5.

Создайте предикат, вычисляющий по натуральному числу N сумму нечетных чисел от 1 до N.

Задание №6.

Реализуйте, используя рекурсию и отсечение, цикл:

а) с постусловием (типа repeat <оператор> until <условие>);

б) со счетчиком (типа for i:=1 to N do <оператор>);

в) со счетчиком (типа for i:=N downto 1 do <оператор>).

Задание №7.

Проверяем равномерность random Пролога.

Создайте предикат, который принимает от человека количество генерируемых чисел и потом генерирует числа (0 или 1). В ходе работы предикат подсчитывает количество 0 и 1 и по итогам генерации выдает результат (сколько 0 и сколько 1).

Задание №8.

Играем с компьютером на количество угаданных выпаданий монеты. Пусть "орел" - 0 и "решка" - 1. Игра продолжается до трех угадываний. На каждом шаге рекурсии программа спрашивает у человека: "орел" или "решка", потом генерирует падение монеты (0 или 1). Если человек угадал, то ему начисляется +1 победа, иначе +1 победа компьютера.