Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Локтионова Оксана Геннадьевна
Должность: проректор по учебной работе
Дата подписания: 06.12.2024 11:25:03
Уникальный программный ключ:
0b817ca911e6668abb13a5d426d39e5f1c11eabbf73e943df4a4851fda56d089

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)
Кафедра программной инженерии

Представление знаний на языке программирования Prolog

методические указания к практическим занятиям для направления подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем

Курск 2024

УДК 004.832.3
Составители: Ю.А. Халин, Е.А. Титенко

Рецензент
Кандидат технических наук, доцент *А.В. Киселев*

Представление знаний на языке программирования Prolog: методические указания к практическим занятиям для направления подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем / Юго-Зап. гос. ун-т ; сост. Ю.А. Халин, Е.А. Титенко. - Курск, 2024. - 27 с. - Библиогр.: с. 27.

Описывается методика программирования задач для интеллектуальных систем с помощью представления знаний на языке исчисления предикатов Prolog. Изложены рекомендации по разработке программ, используя составные объекты, арифметические и логические операции, управление поиском решения, обработку списков, обработку строк, файловые операции.

Методические рекомендации предназначены для студентов, обучающихся по для направления подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем.

Текст печатается в авторской редакции.

Подписано в печать . Формат 60x84 1/16.
Усл.печ. л. 1,57 п.л . Уч.-изд. л. 1,42. Тираж 120 экз. Заказ.
Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94.

Работа 1. ИСПОЛЬЗОВАНИЕ СОСТАВНЫХ ОБЪЕКТОВ

I. ЦЕЛЬ РАБОТЫ

Ознакомление с основными положениями методики использования составных объектов.

II. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Объекты и отношения

Базовыми понятиями для программы на Прологе являются объекты и отношения (факты) между объектами.

Язык Prolog позволяет создавать объекты, содержащие другие объекты. Они называются СОСТАВНЫМИ (сложными) объектами.

Составные объекты дают возможность рассматривать некоторые множества объектов данных как единое целое. Например, дата 1 января 2000 года, состоящая из трех частей, может быть представлена в виде одного составного объекта:

date(1, «января»,2000).

Сложные объекты состоят из функтора и соответствующих ему подобъектов:

functor (object1, object2, … objectN).

Функтор – это просто имя, которое определяет вид составного объекта данных и объединяет его аргументы. Функтор без объектов записывается так: **functor()** или **functor.** Из этого следует, что простой объект является частным случаем составного объекта.

Аргументы составного объекта сами могут быть составными объектами.

Например, «2 сентября 1992 года - день рождения Мишы Кукушкина»:

birthday(person(«Кукушкин», «Миша»),
date(1,«сентября»,1992)).

Все составные объекты можно изображать в виде деревьев (см. рис.1). Корнем дерева служит функтор, ветвями, выходящими из него, - компоненты. Если некоторая компонента тоже является составным объектом, тогда ей соответствует поддерево в дереве, изображающем весь составной объект.

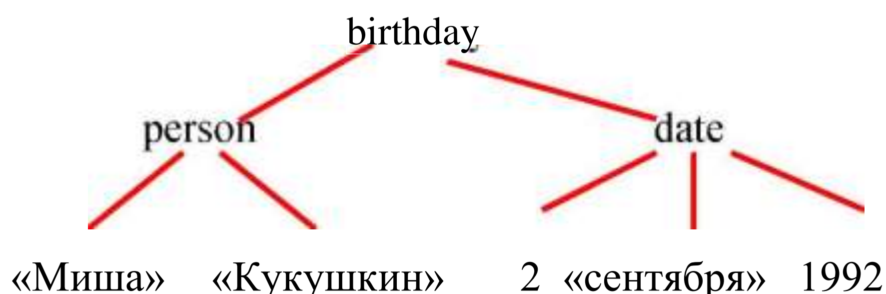

Рис. 1. Пример составного объекта

Все знания выражаются только в виде отношений объектов (предикатов).

Предика́т (лат. praedicatum «заявленное, упомянутое, сказанное») - это утверждение, высказанное о субъекте. В лингвистике субъекту соответствует подлежащее, а предикату - сказуемое. Предикат в программировании - это выражение, использующее одну или более величину с результатом логического типа. В семантике логики предикаты интерпретируются как отношения. Например, в формула будет истинной в интерпретации, если объекты (сущности), обозначенные и , находятся в отношении, обозначенном

Например, такое знание как "John likes Mary" в Прологе выражается как отношение likes, связывающее объекты john и mary (разумеется, порядок перечисления объектов важен, поскольку "John likes Mary" - это совсем не то, что "Mary likes John").

Обычно имена отношений соответствуют глаголам, а имена объектов - существительным.

Факт задает (безусловное) отношение между объектами. Пример факта, выражающего знание "John likes Mary":

likes(john, mary).

Отношение задано предикатом **likes**, а объекты — аргументами этого предиката, в качестве которых выступают константы (имена) **john** и

mary. Аргументы предиката пишутся в скобках, через запятую; в конце ставится точка.

Порядок аргументов задается произвольно, но, будучи единожды выбранным, должен соблюдаться во всей программе. Это значит, что два отношения являются разными, так как имеют различный (!) порядок следования объектов, т.е.

likes(john, mary) ≠ likes(mary, john).

Отношения могу связывать разное количество объектов:

family (mary),
gives(john, book, mary).

В программе, описывающей предметную область, может быть определено несколько отношений (фактов), а один и тот же объект может участвовать в нескольких отношениях:

male(john),
likes(john, mary),
likes(jane, john),
likes(alice, john).

Факты вместе с правилами формируют базу данных, описывающую предметную область.

Целесообразность использования составных объектов заключается в создании иерархии связей между различными объектами и уменьшкении тем самым общего описания предметнойц области. Пример владения различными объектами **Монморенси** приведен ниже.

owner(«боря», «монморенси»).

Здесь оба объекта утверждения **owner** – это простые, структура отношения называется **простой структурой.**

Тем не менее, объект «Монморенси» может быть другом - одноклассником, или собакой, или каким. Для разделения этих случаев можно записать различные утверждения:

owner(«Боря»,friend(«Монморенси»)).
/*У Бори есть друг Монморенси*/
owner(«Боря»,dog(«Монморенси»)).
/*У Бори есть собака Монморенси*/.

Здесь отношение владения объектом – это уже **составная структура.**

Общая структура предиката **owner** может иметь вид

owner (name, article).

Если у Бори есть еще книга «Монморенси» автора Марса Мерседесова, лодка (boat), то типы данных описываются в разделе **domains** так:

article = friend (name); dog (name);
book (name, name1, name2); boat().

Примеры составных объектов из области определения **article** можно использовать в утверждениях, определяющих отношение **owner:**

owner(«Боря», friend(«Монморенси»)).
owner(«Боря», dog(«Монморенси»)).
owner(«Боря»,book(«Монморенси»,«Марс», «Мерседесов»)).

При введении цели:

owner(«Боря», X).

Будет получен ответ:

X = friend(«Монморенси»)
X = dog («Монморенси»)
X = book («Монморенси», «Марс», «Мерседесов»)
X = boat

Таким образом, описание домена в случае использования составного объекта имеет вид:

```
compDom = f1(d1_1, d1_2,..., d1_n);
          f2(d2_1, d2_2,... , d2_m);
          ... ;
          fM(dM_1, dM_2, dM_k),
```

где **fi** - функтор i-ой альтернативы, **{di_j}** - множество компонентов i-й структуры.

Составные объекты могут сравниваться с использованием предиката «равенство» (пример 1.1).

/*Пример 1.1.*/

```
domains
    d = pair(integer, integer); single(integer); none
predicates
    equal(d, d) clauses
    equal(X, X).
```

При вводе следующих целей имеем:
equal(single(1), pair(1,2)). ложно,
equal(pair(3,4)), pair(3,4)). истинно,
equal(none, none). истинно.

Однако, **equal(5, 4).** – дает ошибку, так как имеется несоответствие доменов.

Для удовлетворения последней цели необходимо добавить следующее описание предиката **equal:**

equal(integer, integer),

Тогда **equal(5,4) -** ложно,
equal(5,5) - истинно.

Запрос используется для извлечения новой информации (знаний) из базы данных, т. е. для определения в базе некоторых отношений. В отличие фактов, которые соотносятся с утвердительными предложениями, запрос соотносится с вопросительным предложением. Например, вопрос "Does John like Mary?" может быть

выражен следующим запросом:

?- likes(john, mary)

Служебный символ "**?**"обозначает запрос, а **"likes(john, mary)"** обозначает цель, т. е. то, что Пролог пытается доказать. В остальном простейший запрос не отличается от факта.
Получив запрос, Пролог просматривает базу данных в порядке ввода в поисках фактор удовлетворяющих запросу. Очевидно, что такой факт есть, поэтому ответом будет «**yes**».

Переменная в запросе используется для обозначения неизвестного. Переменные могут использоваться в запросах для перечисления объектов, входящих в некоторое отношение.
Например, запрос

?- likes(john, X).

Даст ответ: **X = mary**

Пусть запрос имеет вид:

likes(X, john).

Ответом будет:

X = jane,
X = alice.

В запросе можно использовать несколько переменных, например:

?- likes(X, Y)

Ответом будет последовательность наборов переменных, разделенных точкой запятой:

X = john,
Y = mary;
X = jane,

Y = john;
X = alice,
Y = john.

Запятые выводятся автоматически.

Вопросы для самопроверки.

1. Структура сложного объекта.
2. Описания домена.
3. Составление простых и составных отношений.
4. Составление запросов.
5. Внутренняя операция унификации переменных.

Работа 2. РЕАЛИЗАЦИЯ АРИФМЕТИЧЕСКИХ И ЛОГИЧЕСКИХ ОПЕРАЦИИ

I. ЦЕЛЬ РАБОТЫ

Изучение способов реализации арифметических вычислений и логических операций в языке программирования Turbo Prolog.

II. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Язык Prolog обладает всеми арифметическими возможностями, присущими другим языкам программирования (Pascal, C). Для этого используется мощный набор математических функций и стандартных предикатов.

Реализация арифметических операций

Выполнение четырех основных арифметических операций с указанием соответствия типов операндов и результатов выполнения приведены в табл. 2.1.

Таблица 2.1.

Операнд 1	Оператор	Операнд 2	Результат
integer	+,-,*	integer	Integer
integer	+,-,*	real	Real
real	+,-,*	integer	Real
real	+,-*	real	Real
integer или real	/	integer или real	Real

Необходимо отметить, что числа можно представлять в различных системах счисления: десятичной или шестнадцатиричной. Последние начинаются со знака доллара **($),** например: **$2EA = 2*16*16+14*16+10 =746.**

В примере 2.1 осуществляется сложение двух чисел, представленных в различных системах счисления.

```
/* Пример 2.1.*/
goal
  write("AA="),readint(AA),nl,
  A=-$2EA+AA, write("A=",A)
```

Математические функции языка приведены в табл. 2.2. Причем аргумент X этих функций является арифметическим выражением.

АРИФМЕТИЧЕСКИЕ ВЫРАЖЕНИЯ, такие, например, как **4, W** или **X = (cos(Y) - 1.5) / 3 + 2.5,** это константа, или переменная, или конструкция, построенная из них путем использования операторов, функций, или скобок и предиката равенства (=).

Таблица 2.2.

Предикат	trunc(X)
abs(X)	X mod Y
exp(X)	X div Y
ln(X)	
lg(X)	
sqrt(X)	
sin(X)	
cos(X)	
tan(X)	
arctan(X)	
random(X)	
random(Y,X)	
round(X)	

Описание
Абсолютное значение числа X
Вычисляется число e в степени X (*e*)
Натуральный логарифм X
Десятичный логарифм (X)
Квадратный корень из X
Тригонометрические функции
(X - число, выражающее угол в радианах)

Арктангенс действительного числа X
Устанавливает X, как псевдослучайное веществен-ное число с равномерным распределением:

0<=X<1
Случайное целое число X с равномерным распределением 0<=X<Y
Округляет значение X
Усекает X (отбрасывает младшие разряды)
Выдает остаток от деления X на Y
Выдает частное от деления X на Y

Использование функции **random(RandomReal)** позволяет получать последовательность величин **RandomReal,** равномерно распределенных в интервале **0<= RandomReal <1.** А для установления правого граничного значения для случайных величин используется функция **randominit(Y).** Функция же **random(MaxValue,Randomint)** выдает последовательность псевдослучайных величин равномерно распределенных в интервал **0<=Randomint<MaxValue.**

При необходимости получить случайные вещественные числа **Z** в диапазоне от **A** до **B** включительно, нужно воспользоваться следующей целью:

random(X), Z = A + X * (B-A+1).

Для получения случайных целых чисел **Z** в диапазоне от **A** до **B** включительно можно воспользоваться целью:

random(Y,X), Z = A + Y +1.

Значение арифметического выражения может быть вычислено только тогда, когда все переменные, находящиеся в правой части, станут конкретизированными. Вычисления осуществляются с учетом приоритета арифметических операций в следующем порядке:

- вычисляются выражения в скобках,
- во всем выражении реализуются слева направо операции умножения и/или деления,
- выполняются слева направо операции сложения и/или вычитания.

В примере 2.2. реализуется вычисление значений арифметического выражения. При A=2 и B=0,3 X будет равен -2,1815545036. Это пример, демонстрирующий программу на языке Prolog, содержащую лишь один раздел (раздел goal). Использование предикатов ввода-вывода будет пояснено ниже.

```
/* Пример
2.2.*/ goal
      write(«A=»),    readint(A),
      nl,             write(«B=»),
```

```
readreal(B), nl, X = -A +
((cos(B) - 1.5) / 3),
write("X=",X).
```

Пусть необходимо написать программу (пример 2.3), реализующую арифметические операции сложения, вычитания, умножения и деления. Величины, над которыми производятся операции имеют тип **real,** т.е. они могут быть представлены в любой форме записи целой или вещественной. Запись самой операции имеет тип **symbol.** Тогда предикат ОПЕРАЦИЯ **(operation)** имеет, например, следующий вид:

operation(symbol, real, real).

В разделе **clauses** описаны правила выполнения операций с выводом результатов вычислений на экран.

```
/*Пример 2.3.*/
predicates
    operation(symbol,real,real
) clauses
    operation(«+»,X,Y):-
        Z=X+Y, write(X, «+»,Y, «=»,Z),nl.
    operation(«-»,X,Y):-
        Z=X-Y, write(X, «-»,Y, «=»,Z),nl.
   operation(«*»,X,Y):-
        Z=X*Y, write(X, «*»,Y, «=»,Z),nl.
   operation(«/»,X,Y):-
        Z=X/Y, write(X, «/»,Y, «=»,Z),nl.
```

Цели внешние и имеют, например, следующий вид: **operation(«/»,5.5,3.4).**

Результат: **5.5 / 3.4 = 1.6176470588.**

operation(«+»,3.4,0.006E2)

Результат: **3.4 + 0.6 = 4.**

Реализация логических операций

Язык Prolog позволяет выполнять бинарные операции отношений между арифметическими выражениями, символами,

строками и идентификаторами. Для этого используются следующие операторы сравнения:

=, <, <=, >, >=, <>(><).

Пример сравнения: **Y + 2 > 8 - X .**

Операция «=» устанавливает соответствие между левой и правой частями выражения. При согласовании переменных действуют следующие правила:

- если X неконкретизированная переменная, а Y - конкретизированная, то X и Y равны,

- целые числа и атомы всегда равны самим себе: например, **613 = 613 , book = book** являются верными утверждениями, а **245 = 45, abc = bcd** - неверными.

- две структуры равны, если они имеют один и тот же функтор и одинаковое число аргументов, причем все соответствующие аргументы равны.

Когда для сравнения вещественных величин используется предикат равенства, нужно позаботиться о том, чтобы приближенное представление вещественных чисел не привело к непредсказуемым результатам. Так цель **4.9999999999 = 5.0000000000** не будет удовлетворена. Это указывает на то, что при проверке на равенство двух вещественных чисел лучше определять, лежит ли их разность в заранее определенных пределах.

Кроме числовых выражений можно также сравнивать простые символы, строки и символические имена, например:

'a' < 'w', /*тип **char*/**

«MISHA» < «MASHINA», /*string*/

X1=misha, X2=masha, X1>X2 /*symbol*/

При этом Prolog преобразует **'a' < 'w'** в выражение **97 < 119,** т.е. сравнивает значения кодов **ASCII,** и отношение истинно. Сравнение **«MISHA» < «MASHIHA »** ложно, т.к. первые отличающиеся символы в сравниваемых строках **I** и **A,** а их коды **73** и **65.** Соответственно сравнение **«bb» > «b»** истинно.

Символические строки нельзя сравнивать непосредственно. В примере **X1=misha, X2=masha** символическое имя **misha** не может быть сравнено непосредственно с **masha** . Они должны быть связаны с переменными или записаны как строки (тип **string).**

В примере 2.4 осуществляется проверка принадлежности целых чисел **Z** интервалу **(X,Y),** где **X** - минимальная граница интервала, а **Y** - максимальная.

```
/*Пример 2.4.*/
predicates
     include(integer, integer,
integer) clauses
     include(X,Y,Z):-
     Z>=X, Z<=Y, nl,
     write(«Число »,Z, « лежит в интервале от »,X,« до », Y,«.»).
```

Вопросы для самопроверки

1. Какое назначение предиката **random(RandomReal)?**
2. Как использовать язык Prolog в режиме калькулятора?
3. Перечислите основные правила установления соответствия между левой и правой частями выражения при выполнении операции «=».

Работа 3. УПРАВЛЕНИЕ ПОИСКОМ РЕШЕНИЯ

I. ЦЕЛЬ РАБОТЫ

Изучение способов реализации рекурсивных зависимостей.

II. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

В языке Prolog используются две конструкции правил, реализующих многократное выполнение задачи. Это ***ПОВТОРЕНИЕ*** и ***РЕКУРСИЯ.***

Структура правила, реализующего повторение, такая:

repetitiverule:-
<предикаты и правила>, fail. /* неудача*/.

Структура правила, реализующего рекурсию:

recursiverule:-
<предикаты и правила>, recursiverule.

В правиле, реализующем повторение, используется встроенный предикат **fail** (неудача). Это всегда ложный предикат, вызывающий откат для поиска других утверждений, которые бы обеспечили вычисление цели. В теле правила рекурсии последним правилом является само правило **recursive_rule.**

Используемая конструкция <предикаты и правила> в теле правил содержит предикаты и правила программы.

МЕТОД ОТКАТА ПОСЛЕ НЕУДАЧИ Для управления вычислением внутренней цели ***ПРИ ПОИСКЕ ВСЕХ ВОЗМОЖНЫХ РЕШЕНИЙ*** используется метод отката после неудачи. Правило

repetitiverule:-
<предикаты и правила>, fail.

будет выполняться столько раз, сколько существует альтернатив для введенной цели.

Для демонстрации использования предиката **fail** и метода ОПН можно привести пример 3.1 о городах. В результате выполнения этой программы выводится небольшой перечень городов северо-запада России, включенных в базу данных.

```
/*Пример 3.1.*/
domains
    name=symbol
predicates
    town(name)
    show_towns
goal
    write("Города северо-запада России:"),nl,
    show_towns.
clauses
    town ("Петербург"). town ("Новгород"). Town
    ("Псков"). town ("Петрозаводск"). town ("Выборг").
    town ("Ивангород"). town ("Ладога"). town
    ("Вологда").
show_ towns:-
    town (Town), write("  ", Town),nl,
fail.
```

На рис. 3.1 изображен процесс отката при реализации примера 3.1. База данных примера содержит 8 альтернативных утверждений для предиката **town(name).** Prolog ищет в базе данных сопоставимые утверждения, при этом внутренние унификационные программы (ВУПы) сначала означивают переменную **Town** объектом первого утверждения **Петербург.** Но существуют и другие утверждения в базе, которые могли бы быть использованы при вычислении подцели **town(Town).** Поэтому для запоминания места возможного отката ВУПы устанавливают указатель отката на следующее утверждение базы данных - **town("Новгород").** Значение **Петербург** переменной **Town** выводится на экран монитора, предикат **fail** вызывает неуспешное завершение правила, переменная **Town** освобождается, а ВУПы осуществляют откат в точку 1 и так далее.

Этот пример показывает применение метода ОПН для извлечения данных из каждого утверждения базы данных при использовании внутренней цели.

Метод отсечения и отката

Для того, чтобы при выполнении программы была возможность ОГРАНИЧИТЬ ПОИСК В БАЗЕ ДАННЫХ, используют средства управления откатом. Это средство называется методом отсечения и отката (метод ОО) и позволяет, используя встроенный предикат **cut,** который обозначается символом восклицательного знака **(!),** осуществлять «фильтрацию данных» по некоторым условиям, т.е. УПРАВЛЯТЬ МЕХАНИЗМОМ АВТОМАТИЧЕСКИХ ОТКАТОВ.

Этот предикат, вычисление которого всегда успешно, заставляет ВУПы заблокировать все указатели откатов, установленные при вычислении предыдущих подцелей. Например, имеется некоторое правило:

pr:-aa,bb,!,cc

и предикату **aa** соответствуют факты и правила **a1, a2, a3;** предикату **bb - bl, b2; cc - cl, c2, c3, c4.**

Если бы отсутствовал предикат **cut,** то было бы предпринято 3*2*4=24 попытки поиска решений. Но пусть осуществляется последовательный поиск решения с откатом до тех пор, пока факт **a1** не удовлетворит предикату **aa,** а **bl** - предикату **bb,** далее происходит успешное завершение предиката **cut.** Затем при переборе фактов, соответствующих предикату **cc,** делается попытка отката на предыдущий предикат **bb,** но **cut** (отсечение) не позволяет это сделать и процесс поиска альтернативных решений в правиле заканчивается.

В связи с отсутствием в языке Prolog конструкций, аналогичных операторам циклов процедурных языков, отличительной его особенностью от большинства языков программирования является широкое использование РЕКУРСИИ.

Рекурсией называется правило, содержащее само себя в качестве компоненты. Обычно Prolog-программа представляет собой совокупность рекурсивных или взаиморекурсивных определений.

ПРОСТАЯ РЕКУРСИЯ

Прежде всего необходимо показать подводные камни, возникающие при использовании рекурсии. Следующее правило

pr:- pr

имеет такой смысл: "**pr** истинно, если **pr** истинно". Непосредственное использование этого правила может привести к бесконечному циклу (бесконечной рекурсии), а это приведет к переполнению стека, что нежелательно, т.к. могут быть потеряны промежуточные результаты вычислений. Ниже приводится пример программы с использованием бесконечной рекурсии:

```
predicates
	write_string
goal
	write_string.
clauses
	write_string:-
		write("Изучайте PDC Prolog!"),nl,
		write_string.
```

При запуске этой программы на экран выдается строка "Изучайте PDC Prolog!", а т.к. последняя компонента правила **write_string** является самим правилом, то опять осуществляется выдача на экран той же строки и т.д. Такая рекурсия называется БЕСКОНЕЧНОЙ и практического применения не имеет.

Рекурсия может стать конечной, если в правило включается УСЛОВИЕ ВЫХОДА. Пусть, например, генерируется последовательность целых случайных величин, равномерно распределенных в интервале от 0 до 9, до тех пор пока не будет получено число 5. В этом случае процесс генерации и выполнение программы завершается (пример 3.2).

```
/*Пример
	3.2.*/
	predicates
		write_number
	goal
		write_number
	. clauses
		writenumber:-
			random(9,Randomint),
			Randomint < > 5,
```

write(« »,Randomint),
write_number.

Первой компонентой правила **write_number** является встроенный предикат **random,** генерирующий последовательность случайных величин. Значение случайной величины присваивается переменной **Randomint.** Следующее подправило проверяет, является ли означенное значение переменной **Randomint** числом 5. Если нет, то подправило успешно, число выводится на экран и производится рекурсивный вызов **write_number.** Процесс продолжается до тех пор, пока не будет сгенерировано число 5. После этого выполнение программы завершается.

ОБОБЩЕННОЕ ПРАВИЛО РЕКУРСИИ Схематически обобщенное правило рекурсии (ОПР) записывается следующим образом:

<имя 1правила рекурсии>:-
<список предикатов>,
<предикат, определяющий условие выхода>, <список предикатов>,
<имя правила рекурсии>,
<список предикатов>.

Принцип работы этого правила идентичен работе правила простой рекурсии. Три конструкции **<список предикатов>** на рекурсию не оказывают влияния. Успешное выполнение предиката, определяющего условие выхода, вызывает продолжение рекурсии, а неуспешное - ее прекращение. Конструкция в теле правила **<имя правила рекурсии>** - это само рекурсивное правило и его успех вызывает (продолжает) рекурсию.

Для получения навыков использования ОПР-метода необходимо рассмотреть еще два примера: генерации (без использования предиката **random)** ряда чисел (пример 3.3) и вычисления факториала (3.4).

Пусть требуется сгенерировать последовательность целых чисел от 1 до 5 (пример 3.3).

/*Пример 3.3.*/
predicates

```
        number(integer)
    goal
        write("Числовой ряд:"),
        number(1),
        write(" . Конец!").
    clauses
        number(6).
        number(Number):
        -
            Number<6,
            write(" ", Number),
            New_number=Number+1,
              number(Newnumber).
```

В рассматриваемом примере первая и последняя конструкции **<список предикатов>** общего правила рекурсии не используются. Именем правила рекурсии является: **number(Number).** В этом правиле предикатом, определяющим, условие выхода, является **Number<6** и, когда **Number=6,** правило успешно, а программа завершается.

Средняя конструкция **<список предикатов>** выводит на экран значение переменной **Number** и затем увеличивает ее на единицу **New_number=Number+1.** Причем для нового числа используется новая переменная **New_number.** И, наконец, программа завершается именем правила рекурсии в теле правила **number(New_number).**

При запуске программы осуществляется попытка вычислить подцель **number(1)** и для этого она безуспешно сопоставляется с утверждением **number(6).** Затем эта подцель сопоставляется с головой правила **number(Number)** и сопоставление успешно, а переменная **Number** принимат значение **1.** Сравнение **1<6** успешно, поэтому условие выхода не выполняется, и значение переменной **Number=1** выводится на экран, а переменная **New_number** становится равной **2.** Затем правило **number(New_number)** со значением параметра **2,** присвоенным переменной **New_number,** вызывает само себя. Несовпадение имен переменных **Number** и **New_number** не имеет значения, т.к. они указывают при передаче значений только на позицию в списке параметров.

Далее продолжается выполнение программы пока **New_number** не станет равным **6,** а это значит, что условие выхода выполняется успешно и программа завершается. Результат на экране такой:

Числовой ряд: 1,2,3,4,5. Конец!

Следующий пример посвящен вычислению факториала

n! = n * (n-1) * (n-2) *...* 3 * 2
* 1. Причем n! = n * (n-1)!, а 1! = 1.
Например 6! = 6 * 5 * 4 * 3 * 2 * 1 = 720.

```
/* Пример
3.4.*/ domains
    n=integer
    f=real
predicates
    factor(n,f)
goal
    write("n="),readint(N),nl,write(N,"!="),
    factor(N,F), write(F),nl.
clauses
    factor(0,1):-!.
    factor(N,F):-
    NN=N-1,factor(NN,PF),F=N*PF.
```

Вопросы для самопроверки

1. Выполнение арифметических операций
2. Выполнение логических операций
3. Объясните назначение предикатов **fail** и **cut.**
4. Как реализуется метод отсечения и отката?
5. Объясните структуру обобщенного правила рекурсии.

Работа 4. ОБРАБОТКА СПИСКОВ

I. ЦЕЛЬ РАБОТЫ

Изучение приемов задания списков и способов реализации операций над списками.

II. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

СПИСОК - это динамическая структура, элементами которой могут быть атомы, списки или структуры. Список является еще одним типом данных наравне с основными типами **char, integer, real, string, symbol** и **file.** Это упорядоченная последовательность равноправных объектов, которая может иметь произвольную длину. Список - это набор объектов одного и того же типа.

Элементом списка может быть атом, переменная, составной терм или список. Основное отличие списков от массивов, используемых в процедурных языках заключается в том, что в списке никогда не задается количество элементов (массив же обычно имеет фиксированную длину), в него не могут входить элементы разных типов данных.

Элементы списка заключаются в квадратные скобки и отделяются друг от друга запятыми. Ниже приведены примеры списков с элементами типа **real, integer** и **symbol:**

[1.4,2.4,3.5,3.6,4.7],
[0,1,2,3,4,5,6],
["Информатика", "Программирование", "Интеллект"].

Важно, что все элементы списка принадлежат к одному типу доменов. Поэтому список

[a,'b',"c"]

некорректен, т.к. состоит из элементов разных типов. Количество элементов списка называется его длиной. Длина списка

["Миша", "Маша", "Вася"]

равна 3. Пустым или нулевым списком называется список, не содержащий элементов. В этом случае он называется атомом и обозначается так: **[].**

Использование списка требует его описания по крайней мере в трех разделах программы. В разделе **domains** описывается используемый тип данных. Например:

domains

town_list=town*/* объявляется список городов*/

town=symbol /* объявляются элементы town списка townlist; их

тип symbol*/

или объявление типа данных может быть таким:

town_list=symbol*.

Если список состоит, например, из списков целых чисел **[[1,2,3],[2,3,4],[3,6,7],[]] ,** то его объявление ничем не отличается от предыдущего: **listList = integer* .** В разделе **predicates** указывается имя предиката и в скобках -имя списка **towns(town_list)** или **numbers(listList).**

В разделе **clauses,** как всегда, приводятся утверждения, т.е. конкретные значения предикатов. Пример 4.1 иллюстрирует описание списка городов.

```
/* Пример 4.1.*/
domains
    town_list=town*
    town=symbol
predicates
    towns(town_list)
clauses
    towns([«Петербург»,«Псков»,«Новгород»,«Ладога»,«Ямбург»]).
```

Реализация такой программы возможна при использовании, например, следующих внешних целей:

towns(All). или **towns([X,_,_,Y,_]).**

Список не имеет никаких предикатов для своей обработки. Для снятия этого неудобства введена единственная операция над элементами списка, называемая **МЕТОДОМ РАЗДЕЛЕНИЯ СПИСКА НА ГОЛОВУ И ХВОСТ.** При этом непустой список рассматривается как структура, состоящая из двух частей:

[Head|Tail].

Переменная **Head** (голова списка) это его первый элемент или фиксированное количество элементов, отделенных символом "|" (вертикальная черта). А переменная **Tail** (хвост) это список оставшихся элементов списка.

В таблице 4.1 приведены примеры деления списка на голову и хвост. При этом видно, что хвост списка - всегда список, а голова - элемент списка, и разделение списка на голову и хвост не зависит от длины списка. Последовательное применение этого метода позволяет легко осуществлять различные операции над списками и написать практически любую программу по обработке любого мыслимого списка.

Таблица 4.1

Список	Голова	Хвост
[1,2,3,4]	1	[2,3,4]
[abcd]	abcd	[]
["Изучайте","PDC","Prolog"]	Изучайте	["PDC","Prolog"]
[A*B,C-D]	A*B	[C-D]
[[E,F,G],h,[i,J]]	[E,F,G]	[h,[i,J]]
[]	Не определено	Не определено

Вопросы для самопроверки.

1. Что такое список и как объявляется список.
2. Метод обработки списка.
3. Голова и хвост списка. Синтаксис разделения списка на 2 части.
4. Рекурсивная обработка списков.

Практические задания

№ п/п	Задача	Параметры
1	Составить базу данных автомобилей и построить запросы поиска по числовым и текстовым полям	Количество моделей автомобилей – 10, Общее число параметров -5
2	Написать программу приветствия, ввода и вывода имен	Количество вводимых имен-3
3	Составить базу данных канцелярских принадлежностей и построить запрос поиска блокнотов, заросы генерации всех элементов в базе данных	Количество типов канц принадлежностей - 10

4	Составить сложные объекты описания генеалогического дерева	Глубина дерева – 5 уровней, Количество потомков – до 3
5	Написать рекурсию ввода и проверки на четность	Верхний предел – число 20
6	Написать рекурсию проверки имени в списке	Длина списка – 10 элементов, Тип элемента списка - text
7	Составить базу данных расписания поездов	Количество направлений -10, Число параметров - 5
8	Составить базу данных классов в школе	Учебные смены -2 , количество уроков – до 6, ФИО классного руководителя
9	Написать программу поиска первого ученика в списке с фамилией «Иванов»	Предусмотреть реакцию на отрицательный результат поиска
10	Составить базу данных «Аптека», построить запрос поиска лекарства по фирмам-производителям	Количество препаратов – 15, число полей – 5 (наименование, цена, производитель, срок годности, вес/объем)

Список литературы

1. Ц. Ин, Д. Соломон Использование ТУРБО-ПРОЛОГА. – М.: Мир, 1988. – 608 с.
2. Марселлус Д. Программирование экспертных систем на Турбо Прологе. - М.: «Финансы и статистика», 1994. – 256 с.