## § 7. ОСНОВЫ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ ПРОЛОГ

## 7.1. ОБЩИЕ СВЕДЕНИЯ

Язык Пролог является представителем семейства языков логического программирования и в сравнении с традиционными языками программирования, предназначенными для записи алгоритмов, такими как Бейсик, Фортран, Паскаль, Си, обладает существенными особенностями:

• программа на Прологе не является алгоритмом, а представляет собой запись условия задачи на языке формальной логики (т.е. это дескриптивный, описательный язык программирования);

• язык Пролог предназначен не для решения вычислительных или графических задач, а для решения логических задач, для моделирования процесса логического умозаключения человека; вычисления же и графические построения выполняются в Прологе как побочный продукт логического вывода;

• Пролог требует особого стиля мышления программиста,что затрудняет изучение его теми, кто уже привык к процедурному программированию, поэтому, так называемые, практические программисты не стремятся переходить на этот язык, что мешает росту популярности Пролога; однако во многих странах (Японии, Англии, Франции, Германии, Израиле и т.д.) расширяется практика применения Пролога в образовании как первого изучаемого языка программирования; переход к процедурным языкам типа Паскаля в этом случае трудностей не вызывает.

Все это позволяет отнести Пролог в существующем делении языков программирования на языки низкого и высокого уровня к языкам сверхвысокого уровня. В японском проекте создания в 90-х годах XX века компьютеров 5-го поколения (обладающих искусственным интеллектом) Пролог положен в основу аппаратной организации и разработки программного обеспечения. Нынешний Пролог, безусловно, не является окончательным вариантом языка программирования ЭВМ 5-го поколения и в ближайшие годы получит существенное развитие. По-видимому, он сыграет роль Бейсика дескриптивного программирования: его значение и возможности в популяризации и распространении идей логического программирования чрезвычайно велики.

Изучению языка Пролог очень способствует предшествующее изучение математической логики, понятийной системой которой он пользуется.

Программирование на Прологе включает в себя следующиеэтапы:

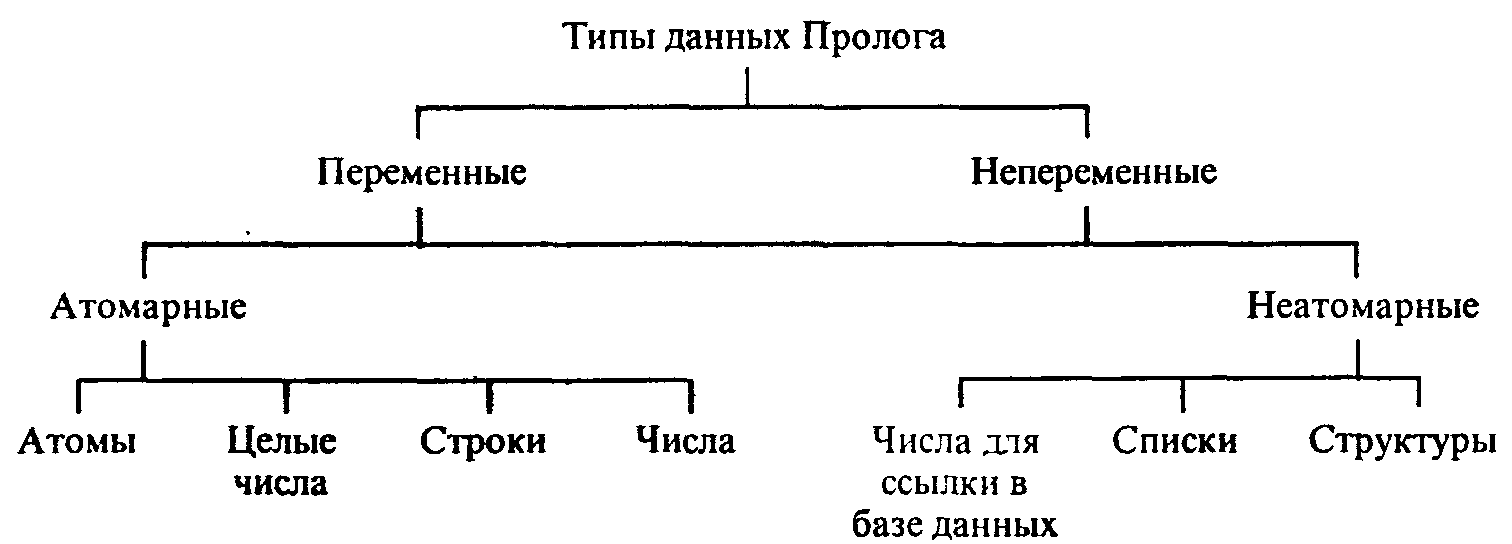
1) объявление фактов об объектах и отношениях между ними;

2) определение правил взаимосвязи объектов и отношений между ними;

3) формулировка вопроса об объектах и отношениях между ними.

Имена - это последовательности букв и цифр, начинающиеся с буквы (строчной !). Системы программирования на Прологе для компьютеров допускают использование лишь латинских строчных и прописных букв: а .. z, A .. Z. Использование русских строчных и прописных букв: а .. я, А .. Я не допускается. При практической работе с интерпретатором рекомендуется, чтобы смысл имен оставался понятным, использовать в качестве имен запись русских слов латинскими буквами. В данном параграфе мы будем записывать все именарусскими буквами, чтобы сделать смысл программ наиболее понятным. При запуске этих программ в «англо-язычных» системах программирования нужно заменять русские буквы в именах на латинские.

Типы данных включают переменные, атомарные значения и структуры (рис. 3.15).



*Рис.3.1 5.* Классификация типов данных Пролога

Примеры *специальных атомов*:

*: - (* обозначающая импликацию),

? (вопрос, обозначающий отрицание),

! (предикат отсечения, рассматривается далее).

***Переменные*** обозначаются последовательностью буквой и цифр, начинающейсяс заглавной буквы. Особый вид переменной - *анонимная переменная \_ ,* используемая в качестве аргумента предиката, когда конкретное значение переменной несущественно.

***Структура*** *-* это конструкция, состоящая из имени структуры и заключенного в скобки списка ее аргументов, разделенных запятыми. Элементами структур могут быть числа, атомы, переменные, другие структуры и списки. Примеры структур: str(A,B,C), носит(юрий,пиджак).

***Списки*** представляют собой объединение элементов произвольных видов, разделенных запятыми и заключенных в квадратные скобки. Списки отличаются от структур тем, что количество элементов может меняться при выполнении программы. Примеры списков: [1,3,5,7], [красный,желтый,зеленый].

Основная операция, выполняемая в языке Пролог, - это операция сопоставления (называемая также унификацией или согласованием). Операция сопоставления может быть успешной, а может закончиться неудачно. Определяется операция сопоставления так:

• константа сопоставляется только с равной ей константой;

• идентичные структуры сопоставляются друг с другом;

• переменная сопоставляется с константой или с ранее связанной переменной (и становится связанной с соответствующим значением);

• две свободные переменныемогут сопоставляться (и связываться) друг с другом. С момента связывания они трактуются как одна переменная: если одна из них принимает какое-либо значение, то вторая немедленно принимает то же значение.

*Примеры: 5* сопоставляется с 5, «имеет» сопоставляется с «имеет», «сергей» не сопоставляется с «юрий», «имеет(сергей,машина)» не сопоставляется с «имеет(сергей, телевизор)», «имеет(сергей,машина)» сопоставляется с «имеет(Х,машина)»,в этом случае переменная Х получает в качестве значения атом «сергей».

***Факты -*** это предикаты с аргументами-константами, обозначающие отношения между объектами или свойства объектов, именованные этими константами. Факты в программе считаются всегда и безусловно истинными и таким образом служат основой доказательства, происходящего при выполнении программы.

*Пример 1.* Факты, описывающие телефонные номера:

телефон(иванов,т561532).

телефон(петров,т642645).

телефон(сидоров,т139833).

Это означает: телефон Иванова - 56-15-32 и т.п. Заметим, что перед цифрами номера идет буква ''т". Она делает номер телефона литерной константой, так как числа 561532,642645, 139833 слишком велики, чтобы быть числовыми константами.

*Пример 2,* Факты, описывающие студентов:

нравится(сергей,рэп).

нравится(юрий,джаз).

носит(сергей,блейзер).

носит(юрий,пиджак).

Это означает: «Сергею нравится рэп», «Юрию нравится джаз» и т.п.

***Правила*** *-* это хорновские фразы с заголовком и одной или несколькими подцелями-предикатами. Правила имеют форму

<голова правила> : - <список подцелей>

где знак : - читается «если», а список подцелей состоит из отдельных подцелей, разделенных знаком «запятая» (читаемым как «и»). Правила позволяют определить новые отношения между объектами на основе уже объявленных с помощью фактов. В качестве аргументов в предикатах правила могут использоваться не только константы, но и переменные. На переменные в правилах действуют кванторы общности, поэтому правила очень концентрированно и лаконично выражают конструкции логического вывода. Так, к фактам примера 2 можно добавить следующее утверждение:

крутойпарень(Х):- нравится(Х,рэп),носит(Х,блейзер).

Это означает «любой Х - крутой парень, если Х нравится рэп и Х носит блейзер». Еще примеры правил:

ест(Х,Y): - пища(Y), любит(Х,Y). («Каждый Х ест любой Y, если Y - пища,

и Х любит Y»)

владелец(А,В) : - купил(А,В). («Любой А есть владелец каждого В, если А купил В»)

В Прологе все предложения программы - факты, правила, вопрос - заканчиваются точкой.

Отметим, что в Прологе вместо оператора присваивания имеется более общий и мощный механизм задания значений переменных. Переменные в Прологе получают свои значения в результате сопоставления с константами в фактах и правилах. До тех пор. пока переменная не получила какого-либо значения, она называется «свободной». Когда переменная примет значение, она становится «связанной». Однако, она остается связанной только в течение времени, необходимого для получения одного ответа на запрос, после этого Пролог «развязывает» ее, возвращается и ищет альтернативные решения. Это очень важный момент: нельзя хранить информацию, задавая значения переменных. Переменные служат частью процесса сопоставления, а не «хранилищем» информации. Область действия переменной -ровно одно предложение (правило или запрос программы).

***Вопрос*** *-* отправная точка логического вывода, происходящего при выполнении программы. На любой вопрос компьютер будет пытаться дать ответ «Да» или «Нет» в зависимости от того. согласуется или нет утверждение, стоящее в вопросе, с фактами и правилами базы знаний. Вопрос, не содержащий переменных, является общим: «имеет ли место факт... ?». Так, например, к базе знаний примера 1 можно поставить вопрос

?-телефон(иванов,т123456).

и ответ будет «Нет», так как константа т123456 не согласуется ни с одним фактом.

Если к базе знаний (пример 3)

нравится(сергей ,рэп).

нравится(юрий,джаз).

носит(сергей,блейзер).

носит(юрий,пиджак).

крутойпарень(Х) : - нравится(Х,рэп),носит(Х,блейзер).

задать вопрос

?-крутойпарень(юрий).

то будет получен ответ «Нет». В самом деле, в результате резолюции утверждение в этом вопросе согласно правилу заменится конъюнкцией утверждений

нравнтся(юрий,рэп), носит(юрий,блейзер).

(переменная Х в правиле получила значение «юрий»). Эти утверждения не согласуются с остальными фактами базы знаний. Для вопроса

? - крутойпарень(сергей).

будет получен ответ «Да», так как в этом случае противоречий при согласовании вопроса и базы знаний не возникает.

Вопрос, в котором имеются переменные, является частным: «длякаких значенийпеременных факт ... имеет место ?». В процессе сопоставлений при выполнении программы переменные получат значения тех констант (конкретизируются), для которых сопоставление запроса, в целом, успешно, и будут выведены на экран. Так, в ответ на вопрос

? - телефон(иванов,Х).

к базе знаний примера 1 на экране появится сообщение Х=т561532 и будет дан ответ «Да».

Если к базе знаний примера 3 задать вопрос в форме

?- крутойпарень(А).

то свободная переменная А в вопросе сопоставляется со свободной переменной Х в правиле и совмещается с ней, т.е. становится одним и тем же. В результате резолюции согласно правилу произойдетзамена

крутойпарень(А)

на

нравится(А,рэп), носит(А,блейзер),

а затем предикат «нравнтся(А.рэп)» успешно согласуется с фактом «нравится(сергей,рэп)>>, и при этом переменная А конкретизируется значением «Сергей»; от вопроса теперь остается «носит(сергей,блейзер)», а в базе знаний имеется соответствующий факт. Ответ: «Да» и на экране появится значение присутствовавшей в вопросе переменной А:

А=сергей.

Отметим, что машина «не понимает» используемых в программе имен: «нравится», «носит», «сергей» и т.д. Мы могли бы вместо них использовать любые другие обозначения. Для интерпретатора Пролога существенны только совпадения и различия имен, а также связи между предикатами, устанавливаемые с помощью конъюнкций и импликаций. Осмысленные имена мы будем использовать только для того, чтобы облегчить чтение и понимание программ самим себе. Однако, в Прологе существуют предопределенные имена (встроенные предикаты), которые позволяют выполнить арифметические операции, сравнения, графические построения, ввод-вывод и другие полезные операции как побочный продукт выполнения программы. Встроенные предикаты Arity-Prolog описаны в справке по системе программирования, вызываемой нажатием клавиши F1.

Аналогичный набор встроенных предикатов имеется в других версиях языка Пролог.

### 7.2. АЛГОРИТМ ВЫПОЛНЕНИЯ ПРОГРАММ НА ПРОЛОГЕ

Факты и правила программы на Прологе являются описанием отношений и связей между объектами некоторой предметной области, т.е. записью условия некой логической задачи, которую предстоит решить. Описанные отношения и связи рассматриваются статически. Такой подход к программе называется декларативным. Порядок следования фактов, правил и подцелей в правилах не влияет на декларативный смысл программы.

Вместе с тем, программу можно рассматривать с точки зрения последовательности сопоставлений, конкретизации переменных и резолютивных выводов, происходящих при ее выполнении. Такой подход называется процедурным. Процедурный смысл программы обязательно должен учитываться при программировании на Прологе. Так, факт можно рассматривать как полностью определенную процедуру, для выполнения которой больше ничего не нужно. Правило

А:-В1,В2,...,Вn.

можно рассматривать как определение процедуры А, утверждающее, что для ее выполнения надо определить Bl, B2, ... , Вn. Процедуры Bl, B2, ... , Вn должны выполняться в определенном порядке - слева направо. Если выполнение очередной процедуры завершается успешно, то происходит переход к следующей процедуре. Если же по какой-либо причине очередная процедура выполняется неуспешно, то происходит переход к следующему варианту описания этой процедуры, и порядок поиска такого варианта в Прологе задан - сверху вниз. Поиск подходящих для согласования фактов и правил в базе знаний происходит последовательно сверху-вниз, и если подходящих фактов не найдено - ответ отрицательный. Эта стратегия согласования называется «сверху-вниз» и «замкнутый мир».

Рассмотрим процесс выполнения программы более подробно на примере.

*Программа 112*

а : - b, с, d.

b : - е, f.

с. d. е. f.

? - а.

Выполнение программы начинается с применения метода резолюций к целевому и одному из предложений программы для получения их резольвенты. Подходящее предложение программы подбирается перебором сверху-вниз так, чтобы сопоставление его заголовка с целевым предложением было успешным. В результате резолюции получается новое целевое предложение и метод резолюции применяется к нему и к другому предложению программы. Процесс продолжается до тех пор, пока не будут согласованы с фактами все возникшие при резолюции подцели, табл. 3.6.

Таблица 3.6

**К процессу выполнения программы на Прологе**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер шага резолюции |  | Целевое предложение | Исходное предложение | Резольвента |
| 1 |  | ?-а. | a:-b,c,d. | ?-b,c,d. |
| 2 |  | ?-b,c,d. | b:-c,f. | ?-e,f,c,d. |
| 3 |  |  | ?-е,f,с,d | e. ?-f,c,d. |
| 4 |  |  | ?-f,c,d. | f. ?-c.d. |
| 5 |  |  | ?-c,d. | c. ?-d. |
| 6 |  |  | ?-d. | d. Пустая |

При выполнении логического вывода, если необходимо, происходит конкретизация переменных. Рассмотрим пример.

*Программа 113*

любит(юрий,музыку).

любит(сергей,спорт).

любит(А,книги):-читатель(А),любопытный(А).

любит(сергей,книги).

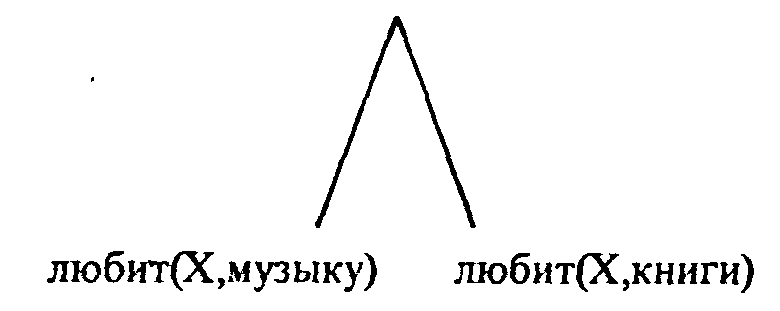
любит(сергей,кино).

читатель(юрий).

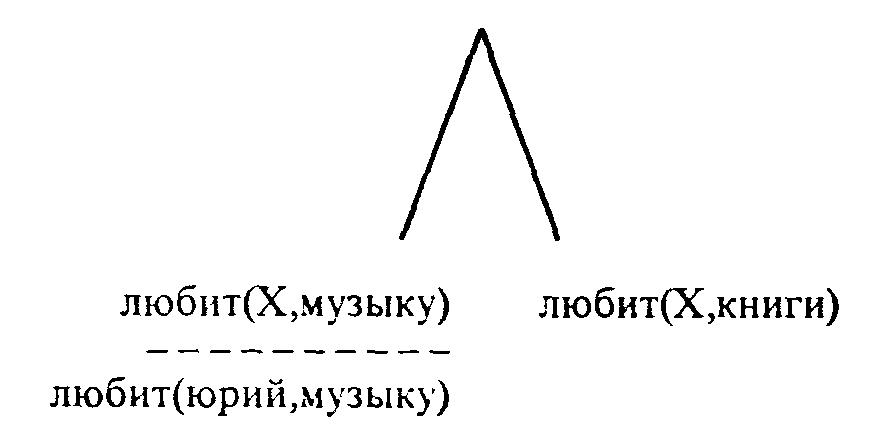
любопытный(юрий).

?- любит(X,музыку), любит(X,книги).

Двойной запрос в этой программе может быть представлен целевым деревом:



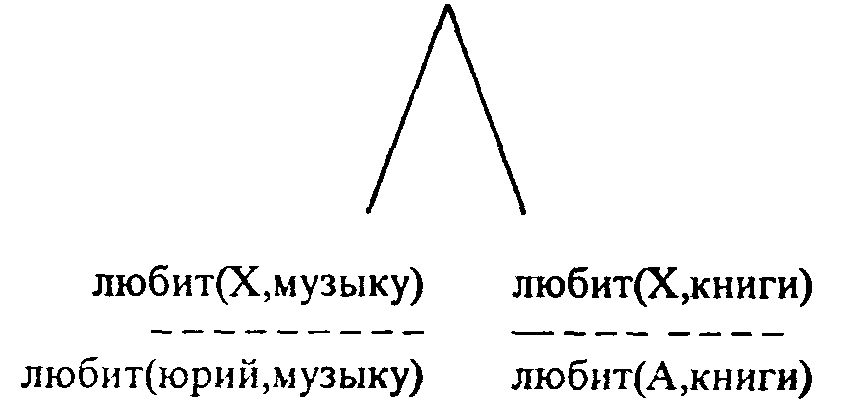
Вначале, просматривая программу сверху вниз. Пролог находит первое предложение, соответствующее первой подцели запроса:

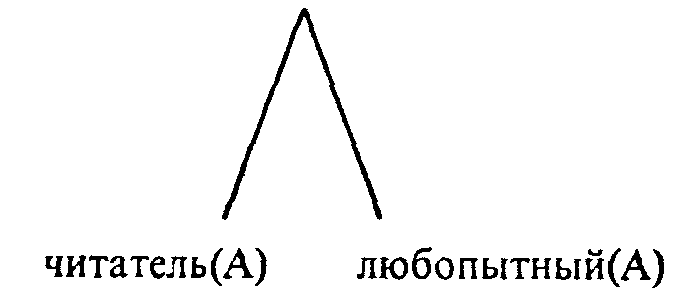


Переменная Х конкретизируется значением «юрий». Начинается согласование 2-й подцели запроса с условием Х=юрий. 1-е и 2-е предложения программы не соответствуют подцели. В 3-ем предложении:

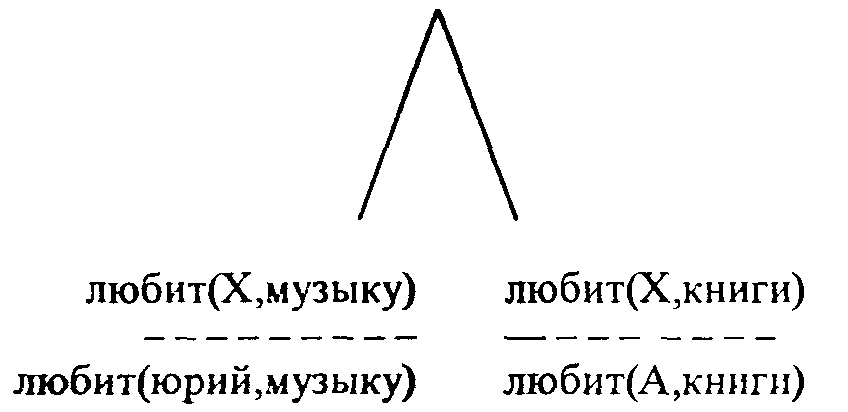
любит(А,книги):-читатель(А), любопытный(А).

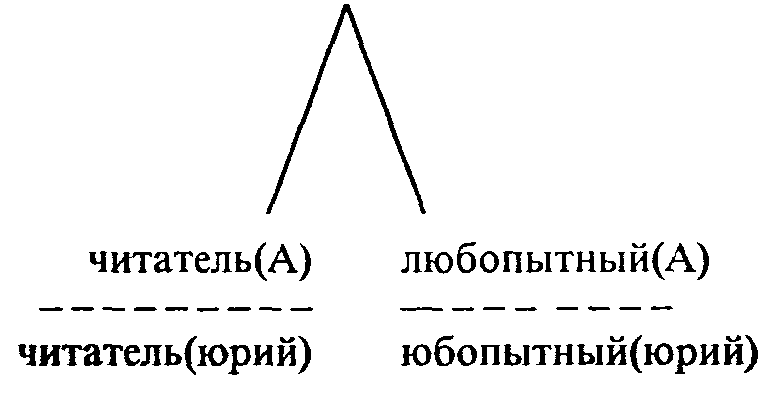
аргумент А заголовка есть переменная, поэтому она может соответствоватьX, т.е. получает значение А=юрин; вторые аргументы совпадают. Теперь тело правила образует новое множество целей для согласования. Получаем целевое дерево:





Затем Пролог будет искать факты, соответствующие новым подцелям. Последнее результирующее дерево:





Рассмотрим еще один пример.

*Программа 114*

любит(оля,чтение).

любит(света,бадминтон).

любит(для,бадминтон).

любит(лена,плавание).

любит(лена,чтение).

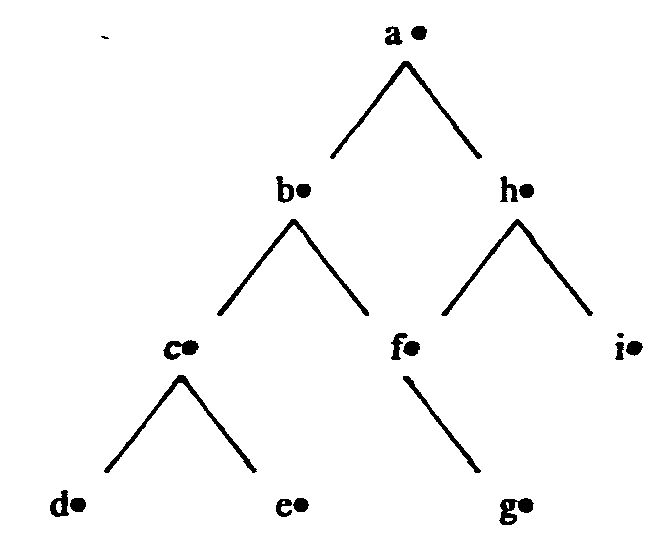
?- любит(X,чтение), любит(X,плавание).

Запрос означает: есть ли люди, которым нравится и чтение, и плавание? Сначала Пролог ищет факт, сопоставимый с первой частью вопроса: любит(Х, чтение). Подходит первый же факт программы

любит(оля,чтение).

и переменная Х связывается значением «оля». В то же время Пролог фиксирует в списке фактов указатель, показывающий состояние процедуры поиска. Далее Пролог пытается согласовать вторую часть запроса при условии Х = оля, т.е. ищет с самого начала программы факт «любит(оля, плавание)». Такого факта в программе нет, и поиск заканчивается неуспешно. Тогда Пролог возвращается к первои части запроса: любнт(Х,чтение) , «развязывает» переменную Х и продолжает поиск подходящих фактов, начиная с ранее установленного в списке фактов указателя Подходит факт «любит(лена,чтение)», переменная Х конкретизируется значением «лена», и далее вторая часть вопроса успешно согласуется с фактом «любит(лена, плавание)». Пролог выполнил в данном примере поиск с возвратом.

Графически процесс выполнения программы представляется в виде обхода бинарного дерева - дерева вывода, типа изображенного на рис.3.16. Вершины дерева обозначают вопросы, а ребра показывают возможные пути вывода, причем для каждого ребра характернысвои правила и унифицирующая подстановка значений переменных.



*Рис.3.16.* Дерево вывода программы на Прологе

Обход дерева начинается с движения от вершины (запроса) по самой левой ветви вниз до конца (abed), при этом запоминаются все точки ветвления (точки возврата). При достижении конца ветви решение будет либо найдено, либо нет. В обоих случаях Пролог продолжает дальнейший поиск решений. Выполняется возврат в последнюю точку ветвления с. При этом конкретные значения, присвоенные переменным при движении вниз на сегменте c-d. отменяются, и движение вниз продолжается по расположенной справа ветви с-е до конца дерева вниз. Затем произойдет возврат в предыдущую точку ветвления b и движение продолжится по ветви bfg, и так до тех пор, пока все дерево вывода не будет пройдено.

### 7.3. РЕКУРСИЯ

Существует целый класс задач, в которых отношения между объектами можно определить, только пользуясь самими определяемыми соотношениями. Получающиеся при этом правила называются рекурсивными.

*Пример:* рекурсивное определение натурального числа:

1) 1- натуральное число;

1. число, на 1 большее натуральногочисла, также натуральное.

В системах логического программирования рекурсия служит также дляописанияциклов, повторений и является важнейшим методом программирования.

Рассмотрим простой пример: вычисление факториала натурального числа n (n!) . Определение n! рекурсивно:

1)1!=1,

2)n!=(n-l)!\*n.

Для описания отношения «факториал» междуn и n! будем использовать двухарный предикат

факт(N,М). Тогда база знаний, соединенная с запросом, приобретает вид (программа 115);

*Программа 115*

факт(1,1).

факт(N,Х): - факт( N-1 ,V), Х is Y\*N.

?- факт(3,А);

В данной программе правило «факт» вызывает само себя - это и есть рекурсия. Запись is Y\*N представляет собой обращение к встроенному предикату «is» («есть») для описания арифметического действия.

Процесс работы программы можно изобразить следующим образом:

?факт(3,A0).

ОТВЕТ: А=6

?факт(1,A2).

Х1= 2\*3 = 6

факт(1,1)

Х2=1\*2=2

Здесь стрелочка вниз означает сопоставление и резолюцию, а стрелочка вверх - возврат и завершение отложенного вычисления.

Правило «факт» вызывает само себя - происходит углубление рекурсии (прямой ход). При этом в памяти ЭВМ выделяется место для переменных А,АО,А1,А2 и N,NO,N1,N2, образующих стеки. При согласовании вопроса с предикатом факт(1,1) рекурсия прекращается и начинается возврат из рекурсии (обратный ход) - выполнение отложенных на прямом ходе согласований. Предикат факт(1,1) играет очень важную роль - это ограничитель рекурсии, условие ее завершения.

Отметим, что Пролог стремится найти все решения поставленной задачи,а значит, после появления ответа А=6 происходит возврат к вопросу ?факт(1,А2) и попытке согласовать его с правилом «факт». Это приводит к бесконечному процессу рекурсии с отрицательными аргументами в «факт», которая завершается при исчерпании глубины зарезервированных интерпретатором Пролога стеков. Ускорить выход из рекурсии можно, добавив к предикату «факт(1,1)» отсечение !:

факт(1,1):-!.

Однако, использование отсечения требует более подробного рассмотрения. В общем случае последовательность предложений в базе знаний не имеетзначения. Однако это не так для рекурсивно-определенных отношений. Например:

родитель(Х):- родитель(Y),отец(Y,Z).

родитель(коля).

отец(коля,петя).

родитель(петя).

В этом случае в первом предложении голова имеет ту же функцию, что и одна из целей - «родитель». В процессе поиска ответа в этой базе знаний будет применено правило: **предложение, стоящее первым, будет применено первым** - известное как ***принцип поиска в глубину****.*

Это приведет к тому, что система будет обращаться только к первому предложению базы знаний и ответ на вопрос не будет найден никогда (образуется бесконечная петля вывода). Однако небольшое изменение базы знаний - перестановка двух предложений местами - приведет к удачному поиску решения.

*Программа 116*

родитель(коля).

родитель(X):- родитель(Y), отец(Y,Х).

отец(коля,петя).

? - родитель(петя).

Неограннчено-повторное обращение к предложению можетбыть и более замаскированным так, как это получается в программе 117.

*Программа 117*

выше(А,В): - ниже(В,А).

ниже(В,А): - выше(А, В).

выше(коля,петя).

?- ниже(петя,коля).

Однако если третье предложение стоит на первом месте, то повторного обращения не произойдет и ответ будет найден.

В общем виде рекурсия на Прологе выглядит так:

Р(1,...).

P(n,...) -Q1,..., Qn, P(n-l,...), R1,... Rm.

Правило Р обращается само к себе, при этом происходит углубление рекурсии. Предикаты Q1, .... Qn выполняются на прямом ходе рекурсии, а R1,..., Rm - на обратном; n - это некоторый условный параметр, входящий в условие продолжения рекурсии, а Р(1,...)- факт, завершающий процесс рекурсии.

Особенно простым случаем рекурсии является простое циклическое повторение. Один из способов организации повторения связан с наличием в базе знаний процедуры вида repeat, repeat: - repeat.

Использование repeat в качестве подцели некоторого правила приводит к многократному повторению остальных подцелей этого правила.

### 7.4. ПРЕДИКАТ ОТСЕЧЕНИЯ И УПРАВЛЕНИЕ ЛОГИЧЕСКИМ ВЫВОДОМ В ПРОГРАММАХ

Управление процессом просмотра предложений является важным аспектом программирования на Прологе. Это осуществляется с помощью специальной встроенной функции «резать», обозначаемой символом "!".

Данная встроенная функция может быть использована длядостижения следующих трех целей:

1) исключения бесконечной петли при выполнении программы;

2) программирования взаимоисключающих утверждений;

1. блокирования просмотра целей.

Продемонстрируем все три случая на примерах.

*Пример 1.* Устранение бесконечных циклов. Обратимся к утверждениям, определяющим последовательность Фибоначчи (числовая последовательность 1, 1, 2, 3, 5, 8,..., в которой каждое число, начиная с третьего есть сумма двух предыдущих).

*Программа 118*

fib (0,\_,1).

fib (1,1,1).

fib (N,G,H) : - fib ( N-l ,F,G), H is F+G.

На запрос

?- fib (0\_ ,F).

получим F = 1, и Пролог сделает попытку сопоставить с запросом второй факт и потерпит неудачу. Однако сопоставление головы третьего утверждения с запросом происходит успешно и осуществляется попытка доказать цель fib(-l,FO,Fl), что, в свою очередь, приводит к цели fib(-2, .., ..) и так далее, т.е. образуется бесконечный цикл.

Однако мы можем устранить такие ситуации, используя отсечение и тем самым указывая Прологу, что не существует других решении в случае успешного согласования граничного условия.

*Программа 119*

fib (0,\_,1) : - !.

fib (1,1,1) : - !.

fib (N,G,H) : - fib ( N-l ,F,G), H is F+G.

Учитывая данное определение fib и задавая вопрос

?- fib(0\_ ,F).

получаем F=l. Других решений нет.

*Пример 2.* Программирование взаимоисключающих утверждений. Процедуру нахождения наибольшегоиз двух чисел можно записать в виде отношения

max(X, Y, М).

Здесь М=Х, если X>=Y, и M=Y, если X<Y. Соответствующие правила таковы:

max(X,Y,X):-X>=Y.

max(X, Y, Y) : - X<Y.

Эти правила являются взаимоисключающими. Возможна более экономная формулировка, использующая понятие «иначе»:

если X>=Y то М=Х иначе M=Y.

На Прологе это записывается при помощи отсечения:

max(X,Y,X):-X>=Y,!

max(X, Y, Y).

*Пример 3.* Блокирование просмотра целей.

*Программа 120*

В.

D

А: - В, С. (1)

С: -D, !, Е. (2)

Е: -F, G, H. (З)

?А.

Говорят, что дизъюнкт (1) «порождает» дизъюнкт (2), так как в правой части (1) есть литера С и эта же литера - в левой части (2). Аналогично дизъюнкт (2) «порождает» дизъюнкт (3). Если (3) неудачен, то в (2) выполнится отсечение: дизъюнкт (2) также считается неудачным, восстанавливается «родительская среда» (1), делается попытка найти альтернативное решение для В. Если бы (2) имело вид С: -D, Е. , то при неудаче в (3) была бы сделана попытка найти альтернативное решение для D.

В других случаях можетбыть необходимым продолжение поиска дополнительных решений, даже если целевое утверждение уже согласовано. В этих случаях можно использовать встроенный предикат fail.

Встроенный предикат fail не имеет аргументов. Он считается всегда ложным.

*Пример:* перебор всевозможных решений.

*Программа 121*

oc(cpm).

ос(msdos).

ос(unix).

печать-всех:-ос(X), write(X), fail.

?-печать-всех.

### 7.5. ОБРАБОТКА СПИСКОВ

На практике часто встречаются задачи, связанные с перечислением объектов. В некоторых случаях при решении задач важно сохранять информацию об уже сделанных шагах решения, чтобы их не повторять. Для решения таких задач в языке Пролог предусмотрены списки.

Список можно задать перечислением элементов. Например, имена учеников класса:

[саша,петя,дима,ксюша,лена].

Элементами списка могут быть не только атомы, но и функции, и вообще любые элементы, даже списки. Заранее длина списка не задается, и в ходе выполнения программы она может меняться.

Альтернативный способ задания списка использует понятия головы и хвоста списка.

Например, в списке [X | Y] Х - это голова списка. Y - его хвост.

Хвост списка по определению также является списком.

Теперь список может быть определен рекурсивно:

1) пустой список [] - список:

2) [X | Y] - список, если Y - список.

Определение списка через его голову и хвост в сочетании с рекурсией лежит в основе большого числа программ, оперирующих списками. Эти программы состоят

1) из факта, ограничивающего рекурсию и описывающего операцию для пустого списка;

2) из рекурсивного правила, определяющего операцию над списком, состоящим из головы и хвоста ( в голове правила), через операцию над хвостом (в подцели).

*Пример I:* определение числа элементов в списке.

*Программа 122*

сколько ([], 0).

сколько ([А|В], N) :- сколько (В, М), N is M+1.

?- сколько ([саша, игорь, лена]), X).

Ответ: Х=3.

*Пример 2:* принадлежность элемента списку.

*Программа 123*

принадлежит (X, [X | Y]).

принадлежит (X, [A |Y ]) : - принадлежит (X,Y).

?-принадлежит (4,(1,3,4,9]).

Ответ:да.

Данная программа имеет очень простой декларативный смысл: элемент принадлежит списку, если он является его головой или принадлежит хвосту списка.

*Пример 3:* соединение двух списков.

Эту задачу можно описать с помощью следующих предикатов:

а) ограничение рекурсии состоит в том, что если к пустому списку [ ] добавить список Р, то в результате получится Р;

б) рекурсия состоит в том, что можно список Р добавить к концу списка [X|Y], если Р будет добавлен к хвосту Y и затем присоединен к голове Х (при этом получается список [Х|Т]).

*Программа 124*

присоединить([ ], Р, Р).

присоединить([XIY], Р, [X | Т]):-присоединить(Y, Р, Т).

? присоединить(L,[джим.R],(джек,бил,джим,тим,джим,боб]).

Ответ:

L=[джек,бил]. К=[тим джим,боб]. L=[джек,бил,джим,тим]. R=[бoб].

Существует традиция использовать для обозначения предикатаслияния двух списков предикативный символ append (по-английски -добавить).

В некоторых случаях постановки вопросов к такого рода программам приходится использовать отсечение (!).

*Программа 125*

append([ ], L, L).

append([A I B] , C, [A | D]):- append(B, C, D).

?-append(X,Y,[1,2]).

Ответ:

X=[]

Y=[l,2]

X=[l]

Y=[2]

X=[l,2]

Y=[].

Если же заменить первое предложение на append([ ], 1,1):- !. и задать тот же вопрос, то получится правильный ответ:

Х=[]

Y=[l,2].

*Пример 4.* удаление элементов из списка.

Программа 126 аналогична проверке принадлежности элемента списку, но требует уже трехарного предиката, один аргумент которого указывает удаляемый элемент, второй аргумент-исходный список и третий - список-результат.

*Программа 126*

удал (X. [X I Y], Y) : - !.

удал (X. [Z I Y], [Z I W]) : - удал (X, Y, W) .

Декларативный смысл: если удаляемый элемент совпадает с головой списка, то результатом программы является хвост списка, иначе удаления производятся из хвоста списка.

Данная программа удаляет первое вхождение в список элемента, связанного с переменной X. Знак отсечения "!"в конце правила предотвращает последующий поиск и вывод лишних вариантов ответов после выполнения ограничительного факта.

Для удаления всех вхождений элемента Х программу надо дополнить:

удал (Х,[ ],[]).

удал (X, [X | Y], W) :- удал (X, Y, W).

удал (X, [Z I Y], W):- удал (X, Y, W).

Декларативный смысл программы таков: пока список не пуст, удалить элемент, если он совпадает с головой списка, значит, отбросить голову списка, а затем удалять его из оставшегося хвоста, иначе надо сразу удалять элемент из хвоста.

*Пример 5:* индексация элементов списка.

Смысл программы 127 состоит в том, чтобы получить элемент под номером N или узнать номер элемента X.

*Программа 127*

получить ([X | Y], 1, X).

получить ([W | Y], N, X) :- N is M+l, получить (Y, M, X).

*Пример 6:* поиск максимального элемента.

*Программа 128*

max ([X], X).

max ([X | Y], X) :- шах (Y, W), X>W, !.

max ([X | Y], W) :-max (Y, W).

Декларативный смысл программы: если в списке один элемент - он и является максимальным, если более одного, то это голова списка, если она больше максимального элемента хвоста, или максимальный элемент хвоста.

*Пример 7:* обращение списка.

Данная задача - самая сложная из рассмотренных. Для ее решения важно сообразить, что обратить список из одного элемента - означает оставить список без изменения. Обратить более длинный список - обратить его хвост, а потом сзади приставить к нему голову исходного списка.

*Программа 129*

обр ([X], [X]) .

обр ([X I Y], Z) :- обр (Y, W), присоединить (W, [X], Z).

В этой программе используется процедура слияния списков, описанная выше.

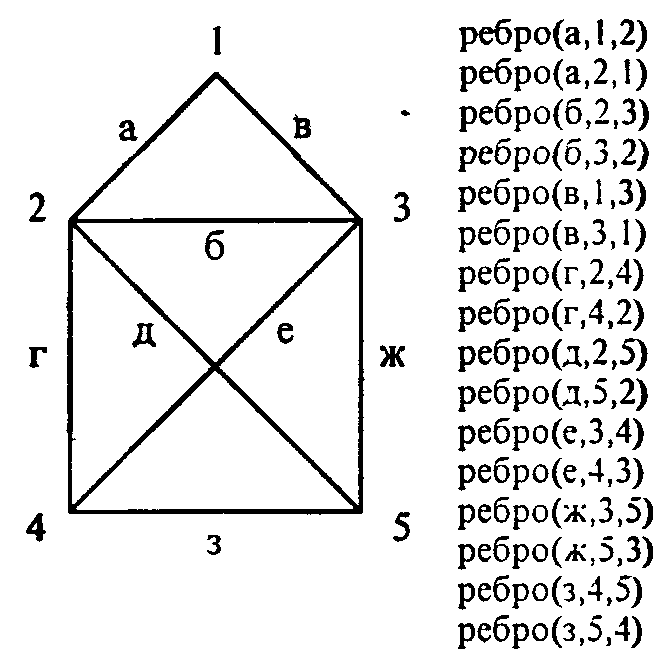
Arity-Prolog располагает значительным числом встроенных предикатов для обработки списков, так что приведенные программы имеют, в основном, учебный характер.

### 7.6. РЕШЕНИЕ ЛОГИЧЕСКИХ ЗАДАЧ НА ПРОЛОГЕ

Целью всего предшествующего изложения была подготовка к данному разделу -решению содержательных логических задач на Прологе, т.е. задач невычислительного характера, в которых особенности Пролога и дескриптивной парадигмы программирования проявляются наиболее ярко.

Рассмотрим пример: нарисовать конверт, не отрывая карандаша от бумаги и не проводя два раза по одной и той же линии.

Введем обозначения, как показано на рис. 3.17. Ребра графа обозначены буквами а, б, в ... (литерные константы), вершины - цифрами 1, 2, 3 ... Опишем структуру графа предикатом вида «ребро (S, А, В)», что означает, что от вершины А к вершине В идет ребро S. Так как граф неориентированный, помимо предикатов вида «ребро (S, А, В)» нужны и предикаты «ребро (S, В, А)». Знания о структуре графа можно представить так, как это записано рядом с рис. 3.17.



*Рис. 3.17.* Задача «конверт»

Решением задачи должен явиться список пройденных ребер графа, причем длина его должна быть равна 8 и в нем не должно быть повторяющихся ребер, что можно описать так:

путь(Т,П) : - длина(П,8), write\_list(П),!.(1)

путь(Т,П) : - ребро(Р,Т,Н),не\_принад(Р,П),путь(Н,[Р|П]).(2)

Переменная Т обозначает текущую вершину графа, а П - список пройденных ребер Правило 1 означает, что если длина списка П пройденных вершин становится равной 8, список П выводится на печать. Это правило ограничивает рекурсивный перебор вершин и ребер, проводимый правилом 2. Правило 2 является генератором перебора, оно перебирает предикаты «ребро()»и находит такое ребро Р из текущей вершины Т в новую Н, чтобы оно не принадлежало списку П, затем это ребро добавляется в качестве головы к списку П, и поиск дальнейшего пути производится уже из новой вершины Н.

Нам потребуется программа, определяющая длину списка,

длина ([],()).

длина ([А | В], N) :- длина (В, М), N is M+1.

а также программа вывода элементов списка на экран

write\_list([]).

write\_list([H | T]):-write(H),write\_list(T).

Задание

?-путь(4,[]).

- искать путь, начиная с вершины 4 и пустого списка пройденных ребер.

*Ответ:* з, ж, в, а, б, д, г, е.

На вопрос ?-путь(1,[]) ответ-«НЕТ».

Аналогично решаются другие задачи, связанные с поиском пути в графе, удовлетворяющего каким-то дополнительным условиям, например задача о коммивояжере. Программа будет состоять

1) из базы знаний о структуре графа - вершинах и связывающихих ребрах(каждому ребру может сопоставляться набор весов);

2) из правил, выражающих дополнительные условия и ограничения на решения задачи и часто связанных с обработкой списков.

3) из рекурсивного правила - генератора перебора ребер и вершин с некоторым ограничивающим предложением, целевым условием;

1. из дополнительных процедур и промежуточных определений.

Интересно, что большинство задач, которые считаются логическими, сводятся к задаче поиска пути в некотором графе - графе состояний задачи. К этому типу задач можно отнести и разнообразные игры. Характерными особенностями многих задач являются следующие:

1) наличие неких дискретных состояний, число которых конечно, и одноиз них принимается за начальное, а другое (или несколько других) за конечное (искомое);

2) определены правила перехода между состояниями;

3) для каждого состояния заданы определенные условия допустимости (оценки) этого состояния.

При анализе предметной области задачи эти состояния, правила перехода и условия допустимости должны быть выявлены, получены соответствующие обозначения и затем записаны с помощью фраз Хорна.

Рассмотрим задачу: имеются два сосуда - на 3 и на 5 литров. Как отмерить с их помощью 4 литра воды ?

В этой задаче состояния связаны с определенным количеством воды V в первом сосуде и W во втором. Начальным состоянием является V=0, \V=0, а конечным V=0, W=4. Переходы между состояниями можно записать в виде правил:

сосуды(V1, W1):- сосуды(V2, W2).

Например, правило

сосуды(0, W) :- сосуды(V, W).

означает, что вся вода из первого сосуда вылита. Обратим внимание на слово «вода» в условии задачи. Для предметной области, связанной с водой, характерно то, что воду можно просто выливать, и данное правило перехода между состояниями допустимо. Если бы задача решалась для молока, то его выливать было бы нельзя, и такое правило было бы недопустимым !

Правило

сосуды(3, W) :- сосуды(V, W). означает, что первый сосуд заполнен полностью.

Не разливая, жидкость можно перелить из одного сосуда в другой только так, что один станет пустым, а другой наполнится. Это можно записать в виде правил

сосуды(3,W):- сосуды(V,W-V+3).

сосуды(V,0):- cocyды(V-W,W).

сосуды(V,5): - cocуды(V-W+5,W).

сосуды(0,W):- сосуды(V,W-V).

При решении данной задачи необходимо также избежать повторения одних и тех же состояний - «переливания из пустого в порожнее». Для этого в предикат «сосуды ( )» следует добавить 3-й аргумент - список пройденных состояний П. Элементы в него будут добавляться парами:

сосуды(V1,W1,[V1,W1|П]):- не\_принад(V1,W1,П), сосуды(V2,W2,П).

Условие, ограничивающее рекурсию, должно иметь вид:

сосуды(\_,4,П) :- write\_list(П).