ПРЕДИСЛОВИЕ

При использовании ЭВМ для решения задач можно выделить два взаимосвязанных способа представления знания:

1) процедурное представление, т. е. определение алгоритма обработки данных;

2) декларативное представление, т. е. определение отдельных понятий, их состояния в конкретные моменты времени и связей между ними.

Традиционные алгоритмические языки (Паскаль, Си, Фортран) являются процедурными, поскольку их цель – описание алгоритма. Но они содержат и декларативные компоненты (описание переменных).

В языке ПРОЛОГ (ПРОграммирование ЛОГическое), напротив, основной является декларативная компонента, так что он предназначен не столько для обработки данных, как для обработки фактов и декларативных правил. Факты представляют собой логические формулы. База знание (БЗ) задается совокупностью таких формул. Логические методы обеспечивают получение новых фактов из фактов, представленных в БЗ.

В процедурных языках программист должен описать для компьютера процесс решения задачи шаг за шагом. В противоположность этому в декларативном языке ПРОЛОГ программист описывает саму проблему и основные правила ее решения, оставляя за системой поиск конкретных действий, приводящих к решению. Поэтому ПРОЛОГ справедливо считается языком более высокого уровня, чем Си или Паскаль. Внешним выражением этого факта является то, что текст программы на ПРОЛОГе может быть в десять раз короче текста программы на Паскале, решающей ту же задачу (хотя, конечно, есть проблемы, неудобные для ПРОЛОГа, но легко описываемые на Паскале или другом процедурном языке).

ПРОЛОГ может использоваться при разработке экспертных систем, а также для следующих задач:

– доказательства теорем и вывода решений в задачах;

– создания пакетов символьной обработки при решении уравнений, дифференцировании, интегрировании и т. д.;

– разработки упрощенных версий систем ИИ;

– создания естественно-языковых интерфейсов для существующих программ;

– перевода текстов с одного языка на другой, в том числе – с одного языка программирования на другой.

ПРОЛОГ появился в 1973 г., когда группа исследователей из Марсельского университета под руководством Алана Колмероэ, создала программу для доказательства теорем, которая была реализована на языке Фортран. Впоследствии этот продукт получил название ПРОЛОГ. Первые годы ПРОЛОГ не находил широкого практического применения. Однако в 1981 г. было объявлено о японском проекте создания ЭВМ пятого поколения, в основе программного обеспечения которых предполагалось использовать логическое программирование. Этот проект создания ЭВМ для обработки знаний вызвал большой резонанс во всем мире. Появились коммерческие реализации ПРОЛОГа, такие как CProlog, Quintus Prolog, Silogic Knowledge Workbench, Turbo Prolog и др.

Наибольшую популярность в нашей стране получила система программирования Turbo Prolog – один из продуктов известной фирмы Borland. В 1988 г. был выпущен Turbo Prolog 2.0, получивший широкое распространение. Фирма Borland распространяла эту версию до 1990 г., а затем компания PDC приобрела права на этот продукт, получивший название PDC Prolog. Позднее был выпущен Visual Prolog, оснащенный стандартными возможностями для быстрого проектирования интерфейса.

В декабре 1997 г. фирма PDC выпустила Visual Prolog 5.0, а с января 1999 г. приступила к распространению версии 5.1. В настоящее время все желающие могут бесплатно скопировать через Internet последнюю версию системы Visual Prolog 5.1 Personal Edition, функционирующую в средах Windows 3.1/95/98, NT, OS/2, SCO UNIX и Linux. Ее загрузочный файл объемом 20 Мбайт можно найти по адресам: http://www.visual-prolog.com/vip/vipinfo /freeware version.htm, http://www.pdc.dk/vip/vipinfo/freeware version.htm. Вариант Personal Edition предназначен для некоммерческого использования, и сообщения об этом имеются во всех приложениях, созданных с его помощью. Кроме того, владельцы Personal Edition не обеспечиваются бесплатной технической поддержкой и на них не распространяются льготы и скидки при приобретении новых версий. В последней версии появились такие усовершенствования, как отладчик, специальный инструментарий и примеры разработки Web-узлов.

Для учебных целей вполне пригоден Turbo Prolog. На использование этой реализации языка рассчитаны все приводимые примеры и задачи.

1. ТЕОРЕТИЧЕСКИЕ ПРИНЦИПЫ ПРОЛОГА

ПРОЛОГ является языком исчисления предикатов. Предикат – это логическая формула от одного или нескольких аргументов. Можно сказать, что предикат – это функция, отображающая множество произвольной природы в множество {ложно, истинно}.

Обозначаются предикаты F(x), F(x, y), F(x, y, z) и т. д..

Одноместный предикат F(x), определенный на предметной области M, является истинным, если у объекта x есть свойство F, и ложным – если этого свойства нет.

Двухместный предикат F(x, y) является истинным, если объекты x и y находятся в отношении F.

Многоместный предикат $F(x_1, x_2, x_3, ..., x_N)$ задает отношение между элементами $x_1, x_2, ..., x_N$ и интерпретируется как обозначение высказывания: "Элементы x_1, x_2, x_N находятся между собой в отношении F".

При разработке логических программ предикаты получают обычно названия, соответствующие семантике описываемой предметной области.

Примеры предикатов:

хищник (X)

супруги (X,Y)

фио (X,Y,Z)

Предикаты, которые нельзя разбить на отдельные компоненты, называют атомарными. Сложные формулы строятся путем комбинирования атомарных предикатов логическими соединителями И, ИЛИ и НЕ.

Одноместный предикат при подстановке в него значения переменной становится «нульместным», т.е. высказыванием-предложением, которое является истинным или ложным:

хищник (тигр).

При подстановке в n-местный предикат конкретного значения его местность становится равной n–1. Таким образом, каждое высказыва-

ние порождается некоторым предикатом, а каждый предикат соответствует множеству высказываний.

Задача ПРОЛОГ – программы заключается в том, чтобы доказать, является ли заданное целевое утверждение следствием из имеющихся фактов и правил.

Решаемая задача представляется в виде совокупности утверждений (аксиом), цель задачи также записывается в виде утверждения. Тогда решение задачи представляет собой выяснение логического следования из аксиом

$$(A_1 \wedge A_2 \wedge A_3 \wedge ... A_n) \rightarrow B.$$

На практике удобно использовать доказательство от противного, т. е. доказывать невыполнимость. Если $A \rightarrow B$ истинно, то $\overline{A \rightarrow B}$ ложно и $A \vee \overline{B}$ ложно.

На этом основан принцип резолюции, который требует представления формул исчисления предикатов в виде набора дизъюнктов, связанных операцией конъюнкции [конъюнктивная нормальная форма (КНФ)].

Правило резолюции имеет вид

$$(X \vee A) \wedge (\overline{A} \vee Y) \rightarrow (X \vee Y).$$

Правило резолюции позволяет соединить две формулы, из одной удалив A, а из другой $\overline{A}$. В этом случае говорят, что A и $\overline{A}$ резольвируют.

Главная идея метода резолюции заключается в проверке того, содержит ли множество дизъюнктов R пустой (ложный) дизъюнкт. При положительном ответе формула, соответствующая R, невыполнима и соответствующее утверждение противоречиво. Таким образом, доказав невыполнимость формулы

$$(A_1 \wedge A_2 \wedge A_3 \wedge ... A_n) \rightarrow \overline{B},$$

можно сделать вывод, что B истинно.

Рассмотрим пример использования метода резолюции, применяя доказательство от противного.

Пусть необходимо доказать истинность выражения

$$((P \vee Q) \wedge (P \rightarrow R) \wedge (Q \rightarrow S)) \rightarrow (R \wedge S).$$

Для использования метода резолюции нужно рассмотреть конъюнкцию совокупности посылок и отрицания заключения в конъюктивной нормальной форме. Для этого выполняется ряд шагов:

1. Приводим посылки к нормальной форме (без $\rightarrow$ и $\leftrightarrow$)

$$P \vee Q;$$

$$\overline{P} \vee R;$$

$$\overline{Q} \vee S.$$

2. Записываем в нормальной форме отрицание заключения

$$\overline{(R \vee S)} = \overline{R} \wedge \overline{S};$$

$$\overline{R};$$

$$\overline{S}.$$

3. Рассматриваем конъюнкцию пяти дизъюнктов

$$\overline{R} \wedge (R \vee \overline{P}) \wedge (P \vee Q) \wedge \overline{S} \wedge (\overline{Q} \vee S)$$

Первые два дизъюнкта дают $\overline{P}$, что в сочетании с третьим дизъюнктом дает Q. Четвертый и пятый дизъюнкты дают $\overline{Q}$.

Таким образом, имеем

$$Q \wedge \overline{Q} = \text{FALSE}.$$

Таким образом, доказано $((P \vee Q) \wedge (P \rightarrow R) \wedge (Q \rightarrow S)) \rightarrow (R \wedge S)$, поскольку противоположное неверно.

Любую программу на ПРОЛОГе можно рассматривать как базу данных. Механизм обработки запросов в ПРОЛОГе называется унификацией. После того как пользователь вводит запрос, интерпретатор приступает к анализу содержимого базы данных, выполняя допустимые подстановки фактов в целевое утверждение, чтобы обосновать его истинность.

2. СТРУКТУРА ПРОГРАММЫ НА ПРОЛОГЕ

Программа на языке ПРОЛОГ включает следующие основные разделы:

описание имен и структур объектов (domains);

описание предикатов – названий отношений, существующих между объектами (predicates);

раздел целевых утверждений (goal), который может отсутствовать; в этом случае программа будет запрашивать целевое утверждение при запуске;

описание фактов и правил, описывающих отношения (clauses).

Имена объектов (констант) в ПРОЛОГе пишутся с маленькой буквы, а переменных – с большой.

Рассмотрим, как можно описать на ПРОЛОГе задачу из некоторой предметной области, например географии.

Программа 1

```
DOMAINS
        gorod, strana = symbol
PREDICATES
        situ(gorod,strana)
CLAUSES
        situ (london, england).
        situ (petersburg, russia).
        situ (kiev, ukraine).
        situ (pekin, asia).
        situ (warszawa, poland).
        situ (berlin, europe).
        situ (X, europe):- situ (X, russia).
        situ (X, europe):- situ (X, poland).
```

Выражение

situ(kiev, ukraine)

описывает тот факт, что город Киев находится на Украине. Ранее в программе был введен соответствующий предикат, работающий с объектами символьного типа. Название города и страны здесь являются константами, поэтому по правилам ПРОЛОГа их нужно писать с маленькой буквы. Вместо блока определений:

```
DOMAINS
        gorod, strana = symbol
PREDICATES
        situ(gorod,strana)
```

можно было просто описать предикат:

```
PREDICATES
        situ(symbol, symbol)
```

Иногда первый вариант предпочтительнее, поскольку позволяет оценивать семантический смысл аргументов предиката.

В конце раздела clauses *Программы 1* описаны два правила. Правило задает новый предикат через предикаты, определенные ранее. Правило

состоит из головы (предиката) и тела – последовательности предикатов. Голова отделяется от тела значком :- , который можно интерпретировать как слово «Если». Таким образом, заключение является головой правила, а тело правила состоит из набора посылок.

Использование правил является основным способом представления знаний в интеллектуальных системах.

Смысл правила состоит в том, что цель, являющаяся головой, будет истинной, если интерпретатор ПРОЛОГа сможет показать, что все выражения (подцели) в теле правила являются истинными.

В правилах буква X (или любая другая заглавная буква, или любое слово, начинающееся с заглавной буквы) обозначает переменную, которая может принимать разные значения.

Так, правило

situ (X, europe):- situ (X, poland)

означает, что любой польский город является одновременно европейским городом. Добавлением новых правил можно пополнять и модифицировать описание задачи. Если мы хотим описать тот факт, что все города Франции являются одновременно европейскими городами, то достаточно добавить всего одно правило

situ (X, europe):- situ (X, france)

и можно будет по-прежнему использовать все остальные факты о городах Европы.

В ПРОЛОГе можно использовать составные объекты. Составные объекты позволяют описывать иерархические структуры, в которых описание одного предиката включает в себя описание других предикатов.

Например:

Программа 2

```
DOMAINS
    personal_library=book(title,author,publication)
    publication= publication(publisher,year)

    collector,title,author,publisher=symbol
    year=integer
PREDICATES
collection(collector, personal_library)
```

```
CLAUSES
    collection(“Иванов”,book(“Война и мир”, “Лев Толстой”,
publication(“Просвещение”,1990))).
```

При описании правил часто возникает необходимость использовать логические связки И и ИЛИ. В качестве связки И используется запятая, а в качестве связки ИЛИ – точка с запятой. Например:

```
gigant(X) :- rost(X,Y),Y>200.

star_or_mlad(X) :- X>70; X<10.
```

ПРОЛОГ имеет большое количество встроенных предикатов, т.е. предикаты, определяемые автоматически. Например, встроенный предикат nl вызывает перевод строки, а встроенный предикат write применяется для вывода информации на экран. Встроенные предикаты используются так же, как и определяемые пользователем предикаты, но встроенный предикат не может являться головой правила или появляться в факте.

Часто используемыми встроенными предикатами являются = (унификация) и логическое отрицание not. Например:

```
student(X) :- X=”Петров”; X=”Иванов”.

xor_student(X) :- not(X=”Петров”), not(X=”Иванов”).

planeta(X) :- not(X=”солнце”).
```

Утверждение not(X = Y) эквивалентно X<>Y.

Иногда бывает полезно использовать предикаты, про которые заранее известно, истинны они или ложны. Для этих целей используют предикаты true и fail. Предикат true всегда истинен, в то время как fail всегда ложен. Последний предикат используется для управления процессом решения задачи на ПРОЛОГе.

ПРОЛОГ-программа может использовать комментарии, которые не влияют на выполнение программы, но могут оказать помощь человеку, читающему программу. ПРОЛОГ игнорирует произвольное число строк, заключенное между символами /* и */. Все, что находится между % и концом строки, также рассматривается как комментарий:

```
/* Здесь записан
        комментарий */
% Это тоже комментарий
```

При описании конкретной предметной области обычно имеется набор исходных фактов и правдоподобных допущений, на основании которых формулируются правила.

Рассмотрим, каким образом на ПРОЛОГе можно описать задачу о семейных отношениях.

Пусть имеются факты об отцовстве:

1) Иван – отец Игоря.

2) Иван – отец Сидора.

3) Сидор – отец Лизы.

Введем также три предиката:

Мужчина (x), означающий, что x – мужчина,

Единокровный (x,y), означающий единокровность x и y,

Брат (x,y), означающий, что x брат y.

Справедливы, очевидно, следующие правила:

1) «Все отцы – мужчины».

2) «Если у детей один отец, то они единокровны».

3) «Брат – это единокровный мужчина».

Рассмотрим вывод решения при ответе на вопрос:

«Есть ли братья у Игоря?».

Программа 3

```
DOMAINS
        person = symbol
PREDICATES
        otec(person,person)
        man(person)
        brat(person,person)
CLAUSES
        man(X):-otec(X,_).
        brat(X,Y):-otec(Z,Y),otec(Z,X),man(X),X<>Y.
        otec(ivan,igor). otec(ivan,sidor). otec(sidor,lisa).
```

Во втором правиле программы указано условие X<>Y. Это позволяет описать ПРОЛОГ-программе тот факт, что человек не может быть собственным братом.

После запроса

```
goal: brat(igor,X)
```

Система выдаст

```
X = sidor
```

Это отвечает нашим представлениям о правильном решении.

Приведенные примеры примитивны, но они позволяют представить неожиданность и полезность решений, которые может сгенерировать ПРОЛОГ при большом количестве фактов и правил в сложной предметной области.

3. ОПИСАНИЕ АРИФМЕТИЧЕСКИХ ОПЕРАЦИЙ

В языке ПРОЛОГ используется ряд встроенных функций для вычисления арифметических выражений, некоторые из которых перечислены в табл. 1.

Таблица 1

Обозначение	Тип операции
>,<,=,>=,<=,<>	Операции сравнения
+, -, *, /	Арифметические операции
X mod Y	Остаток от деления X на Y
X div Y	Частное от деления X на Y
abs(X)	Абсолютная величина числа X
sqrt(X)	Квадратный корень из X
sin(X), cos(X), tan(X), arctan(X)	Тригонометрические функции
exp(X)	Возведение в степень X
log(X)	Десятичный логарифм (ln) числа X
ln(X)	Натуральный логарифм числа X

Для описания любых операций арифметики можно также использовать собственные предикаты. Например:

Программа 4

```
PREDICATES
        add(integer,integer)
        fadd(real,real)
        maximum(real,real,real)
CLAUSES
        add(X,Y):-Z=X+Y,write("Sum= ",Z),nl.
        fadd(X,Y):-Z=X+Y,write("FSum= ",Z),nl.
        maximum(X,X,X).
        maximum(X,Y,X):- X>Y.
```

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

Предикаты ПРОЛОГА не могут появляться в арифметических выражениях. Если требуется, например, переменной R присвоить значение, равное большему из двух выражений X и Y, умноженному на 3, то, используя предикат maximum, это можно записать так:

maximum(X,Y,Z), R= 3*Z.

Вычислить длину гипотенузы по длинам катетов прямоугольного треугольника можно с помощью конструкции:

gipotenuza(X,Y,Z):- Z = sqrt(X*X + Y*Y).

4. ЗАПРОСЫ К ПРОЛОГ-ПРОГРАММЕ

Запрос – это последовательность из одного предиката или множества предикатов, разделяемых запятыми (связка and) и завершающаяся точкой. С помощью запросов можно установить истинность соответствующего выражения. Предикат запроса называется *целью (goal).*

Простые запросы, не содержащие переменных, называют да-нет-вопросами. Они допускают лишь два возможных ответа: “Yes” или “No”. В случае ответа “Yes” говорят, что запрос завершился успехом, цель достигнута.

Например, если *Программу 1* запустить на решение, то она выдаст сообщение

goal: {цель},

что означает готовность к вводу задачи, которая, например, может быть такой

goal: situ(petersburg, europe).

Ответом на этот вопрос будет:

Yes,

т. е. истинно (хотя в явном виде этого факта описано не было).

Использование переменных в запросах позволяет задавать более сложные вопросы. Запросы с переменными могут иметь более одного решения. Первым всегда выводится то из решений, которое найдено первым. Сообщение “No” здесь также говорит об отсутствии очередного решения.

Если ввести запрос к *Программе 1*, иначе:

goal: situ(X, europe),

В ответ на этот запрос будет получено несколько ответов:

X = london

X = petersburg

X = kiev

X = warszawa

В ПРОЛОГ-программе можно использовать и количественные сведения. Это можно сделать так:

Программа 5

```
DOMAINS
    nazvanie,stolica = symbol
    naselenie = integer
    territoria = real
PREDICATES
    strana(nazvanie,naselenie_mln,territoria,stolica)
CLAUSES
    strana(kitai,1200,9597000,pekin).
    strana(belgia,10,30000,brussel) .
    strana(peru,20,1285000,lima) .
```

Если для этой программы поставить задачу следующим образом

goal: strana(X, _,Y, _), territoria > 1000000

То система выдаст значения, удовлетворяющие заданным ограничениям

X = kitai, Y = 9597000

X = peru, Y = 1285000

В этом запросе значки _ применяются для обозначения *анонимных переменных*, значения которых безразличны для получения решения.

Таким образом, использование анонимных переменных не только упрощает процесс поиска решения, но и сокращает объем информации, получаемой пользователем.

В программе на ПРОЛОГе цель может указываться в явном виде в разделе goal. Например:

Программа 6

```
PREDICATES
    hello
GOAL
    hello.
```

```
CLAUSES
    hello:-write("hello").
```

Аргументом встроенного предиката write может являться любой допустимый терм ПРОЛОГА. В случае, когда аргументом является переменная, будет напечатано ее значение. Использование этого предиката позволяет получать в запросе более подробную информацию о решении.

Таким образом, запросы к ПРОЛОГ–программе могут происходить двумя способами – автоматически, при указании цели в разделе goal программы, либо при реализации диалога с пользователем.

В процессе диалога часто бывает необходимо использовать ввод информации с клавиатуры. Для этого имеется набор встроенных предикатов ввода:

```
readln(X)          /* ввод строки */
readchar(X)        /* ввод символа */
readint(X)         /* ввод целого числа */
readreal(X)        /* ввод действительного числа */
```

Язык Turbo Prolog имеет также богатый набор встроенных предикатов для управления текстом, графикой, звуком, и т. д., но эти возможности достаточно традиционны. Для ознакомления с ними можно воспользоваться HELP-файлом программы.

5. УПРАВЛЕНИЕ ПРОЦЕССОМ РЕШЕНИЯ ЗАДАЧИ

Использование предиката fail

В ПРОЛОГе реализован механизм поиска с возвратом (backtracking), при котором система пытается отыскать все возможные решения задачи. Механизм вывода программы запоминает те точки процесса унификации, в которых не были использованы все альтернативные решения, а затем возвращается в эти точки и ищет решение по иному пути.

Однако поиск с возвратом выполняется автоматически только в тех случаях, когда программа решает задачу в результате диалога с пользователем. Если же цель указана в разделе goal программы, то поиск оканчивается после нахождения первого решения задачи. В этом случае для вывода всех решений используется предикат fail.

Предикат fail называют откатом после неудачи. Он вызывает искусственное неуспешное завершение поиска, что позволяет получить все возможные решения задачи.

Программа 7

```
PREDICATES
        gorod(symbol)
        show
GOAL
        write("Это города:"),nl,show.
CLAUSES
        gorod("москва"). gorod("минск").
        gorod("киев"). gorod("омск").
        show :- gorod(X), write(X),nl,fail.
```

После запуска будут напечатаны все названия городов. Проверьте, как будет работать программа без предиката fail.

Использование предиката cut

Предикат отсечения cut обозначается с помощью символа **!**. Он позволяет получить доступ только к части данных, устраняя дальнейшие поисковые действия. В общем виде можно записать, например:

R : – A,B, !, C

Это означает, что если для целей A и B найдено хотя бы одно решение, то дальнейший перебор возможных вариантов значений A и B не нужен. *Программы 8* и *9* иллюстрируют действие предиката cut.

Программа 8

```
DOMAINS
        person=symbol
PREDICATES
        deti(person)
        show
        make_cut(person)
GOAL
        write("Это мальчики:"),nl,show.
CLAUSES
        deti("Петя").deti("Вася").deti("Олег").
        deti("Маша").deti("Оля").deti("Наташа").
        show :- deti(X),write(X),nl,make_cut(X),!.
        make_cut(X) :- X="Маша".
```

Программа 8 напечатает только имена мальчиков.

Программа 9

```
PREDICATES
        buy_car(symbol,symbol)
        car(symbol,symbol,integer)
        color(symbol,symbol)
CLAUSES
        buy_car(Model,Color) :-
        car(Model,Color,Price),color(Color,”светлый”),!,Price<25000.

        Car(“москвич”,”синий”,12000).
        Car(“жигули”,”зеленый”,26000).
        Car(“вольво”,”синий”,24000).
        Car(“волга”,”синий”,20000).
        Car(“ауди”,”зеленый”,20000).
        Color(“синий”,”темный”).
        Color(“зеленый”,”светлый”).
```

Эта программа не найдет ни одного решения, поскольку после дорогих зеленых «жигулей» поиск заканчивается, и более дешевые «ауди» не будут найдены.

6. ИСПОЛЬЗОВАНИЕ РЕКУРСИИ В ПРОЛОГЕ

Правило является рекурсивным, если содержит в качестве компоненты само себя. Рекурсия допустима в большинстве языков программирования (например, в Паскале), но там этот механизм не является таким важным, поскольку имеются другие, свойственные процедурным языкам, механизмы – циклы, процедуры и функции.

Рассмотрим преимущества использования рекурсии на примере. Пусть имеются следующие факты о том, какая валюта котируется выше:

```
doroje(dollar, rubl).
doroje(evro, rubl).
doroje(rubl, iena).
doroje(funt, euro).
```

Выполним запрос:

```
? doroje(evro, rubl).
Yes
```

Будет получен утвердительный ответ, поскольку такой факт явно описан в программе. Если же сделать запрос,

? doroje(evro, iena).

То ответ будет отрицательный, поскольку такой факт отсутствует. Аналогичным будет ответ на вопрос:

? doroje(funt, rubl).

Избежать противоречий здесь можно введением правила, в котором допустимо сравнение между собой не только двух, но и трех объектов:

```
doroje1(X, Y):- doroje(X, Y).                    /* два объекта */
doroje1(X, Y):- doroje(X, Z), doroje(Z, Y).      /* три объекта */
```

Второе правило описывает, очевидно, вариант, когда X>Z, а Z>Y, откуда делается вывод, что X>Y.

Однако цепочка взаимных сравнений может быть длинной. Например, при четырех сравнениях потребуется конструкция:

```
doroje2(X, Y):- doroje(X, M), doroje(M, K), doroje(K, Z), doroje(Z, Y).
```

Описывать такие длинные правила неудобно. Здесь выгоднее применить рекурсию, обратившись к правилу из самого этого правила:

```
doroje1(X, Y):- doroje(X, Y).
doroje1(X, Y):- doroje(X, Z), doroje1(Z, Y).
```

Первое предложение в этой конструкции определяет момент прекращения рекурсивных вызовов.

Второе правило описывает возможности рекурсивных вызовов, когда существуют непроверенные варианты решения.

Вообще, любая рекурсивная процедура должна содержать хотя бы одну из двух компонент:

1. *Нерекурсивную фразу,* определяющую правило, применяемое в момент прекращения рекурсии.

2. *Рекурсивное правило,* первая подцель которого вырабатывает новые значения аргументов, а вторая – рекурсивная подцель – использует эти значения.

База правил просматривается *сверху вниз.* Сначала делается попытка выполнения нерекурсивной фразы. Если условие окончания рекурсии не указано, то правило может работать бесконечно. Например:

write_string :- write(“*****”),nl, write_string.

будет бесконечно печатать звездочки на экране компьютера.

Следующая программа печатает на экране компьютера цифры от 1 до 7.

Программа 10

```
DOMAINS
    number=integer
PREDICATES
    write_number(number)
GOAL
    write(“Это числа:”), nl, write_number(1).
CLAUSES
    write_number(10).
    write_number(N) :- N<10,write(N),nl,N1:=N+1, write_number(N).
```

В разделе clauses даны два описания предиката write_number**.** Если в процессе решения первое описание неуспешно, то используется второе описание.

Программа 11 печатает сумму всех цифр введенного с клавиатуры числа.

Программа 11

```
PREDICATES
    summa(integer,integer)
CLAUSES
    summa(X,Y):-X<10,Y=X,!.
    summa(X,Y):-X1=X div 10,summa(X1,Y1),Z=X mod 10,Y=Y1+Z.
```

Использование предиката ! в описании нерекурсивного правила позволяет избежать здесь переполнения стека.

Существуют проблемы, в которых использование рекурсии особенно выгодно.

Рассмотрим задачу «Ханойская башня» которую, как говорят, придумал в 1883 г. французский математик Люка.

Имеются три стержня, на одном из которых помещены N колец разного диаметра, при этом, чем меньше диаметр кольца, тем выше оно лежит (рис. 1). Например, для четырех колец получается картинка:

Требуется переместить диски с первого на третий стержень за некоторую последовательность ходов, каждый из которых заключается в пе-

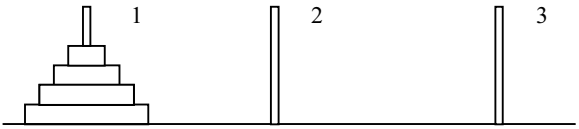

Рис. 1

рекладывания верхнего диска с одного из стержней на другой стержень. При этом больший диск никогда нельзя ставить на меньший диск.

Если ввести обозначения:

<a,b> элементарная операция-перемещение диска со стержня с номером a на стержень b,

(m,a,b) программа перемещения m дисков с a на b.

(1,a,b) → <a,b> перемещение одного диска – элементарная операция.

Очевидно можно записать:

$$(m,a,b) \rightarrow (m\text{-}1,a,c)\ \langle a,b\rangle\ (m\text{-}1,c,b)$$

Т. е. для перемещения m-дисков с a на b нужно:

1) Переместить m-1 – диск с a на c (c – вспомогательный стержень).

2) Переместить нижний диск с номером m с a на b.

3) Переместить m-1 – диск с c на b (c – вспомогательный стержень).

Здесь возникает рекурсия – целевое действие определяется через промежуточные действия того же вида.

Например, пусть m = 3, т. е. имеется три кольца. Тогда:

$$(3,1,3)\rightarrow(2,1,2)\langle 1,3\rangle(2,2,3)$$

Можно переписать в виде

$$(3,1,3)\rightarrow \quad (2,1,2) \quad \langle 1,3\rangle \quad (2,2,3)$$
$$\rightarrow \quad (1,1,3)\langle 1,2\rangle(1,3,2) \quad \langle 1,3\rangle \quad (1,2,1)\langle 2,3\rangle(1,1,3)$$

Окончательно:

$$\langle 1,3\rangle\langle 1,2\rangle\langle 3,2\rangle\langle 1,3\rangle\langle 2,1\rangle\langle 2,3\rangle\langle 1,3\rangle$$

Таким же образом может быть получена программа для любого числа колец.

Существует «восточная легенда» (которую, как говорят, придумал тот же математик Люка), согласно которой в одной из пещер Гималаев три буддийских монаха решают эту задачу для 64 колец. Когда они переложат все кольца, наступит конец света.

Решение задачи требует 2^n-1 действий. Если считать, что одно действие выполняется за 1 с, то уже для 40 колец должно потребоваться более 2000000 лет, что звучит достаточно оптимистически.

Программа 12 решает задачу «Ханойская башня» на ПРОЛОГе.

Программа 12

```
DOMAINS
    loc =right;middle;left
PREDICATES
    hanoi(integer)
    move(integer,loc,loc,loc)
    inform(loc,loc)
GOAL
    hanoi(5).
CLAUSES
    hanoi(N):- move(N,left,middle,right).
    move(1,A,_,C):- inform(A,C),!.
    move(N,A,B,C):- N1=N-1, move(N1,A,C,B), inform(A,C),
move(N1,B,A,C).
    inform(Loc1, Loc2):- nl,write("Диск с", Loc1, " на ", Loc2).
```

7. ИСПОЛЬЗОВАНИЕ СПИСКОВ

Списки – одна из наиболее часто употребляемых структур в ПРОЛОГе. Список – это набор объектов одного и того же типа. При записи список заключают в квадратные скобки, а элементы списка разделяют запятыми, например:

```
[1,2,3]
[1.1,1.2,3.6]
["вчера","сегодня","завтра"]
```

Элементами списка могут быть любые термы ПРОЛОГа, т.е. атомы, числа, переменные и составные термы.

Каждый непустой список может быть разделен на голову – первый элемент списка и хвост – остальные элементы списка. Это позволяет всякий список представить в виде бинарного дерева (рис. 2).

У списка, состоящего только из одного элемента, головой является этот элемент, а хвостом – пустой список.

Для использования списка необходимо описать предикат списка.

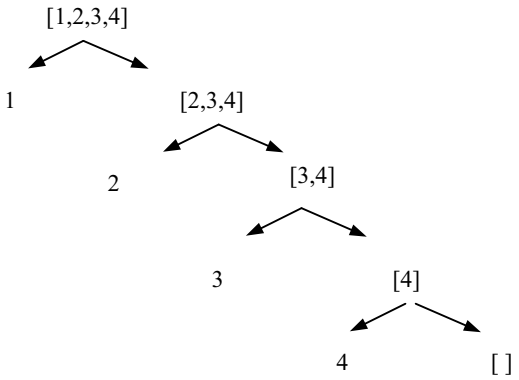

Рис. 2

Например: num([1,2,3]).

Пусть мы хотим описать с помощью списка породы собак.

Программа 13

```
DOMAINS
    dog_list= symbol* /* здесь «*» указывает на список*/
PREDICATES
    dogs(dog_list).
CLAUSES
    dogs(["лайка","борзая","дог","болонка"]).
```

После запроса:

```
goal:    dogs(X)
```

будут напечатаны все породы собак, а после запроса:

```
goal:    dogs(_,_,_,X)
```

получим: X = болонка.

Операция разделения списка на голову и хвост обозначается с помощью вертикальной черты:

```
[Head|Tail]
```

С помощью этой операции можно реализовывать рекурсивную обработку списка. Например, если мы хотим напечатать все элементы списка из *Программы 13* построчно, то это можно выполнить с помощью рекурсивного определения предиката:

```
print_list([ ]).
print_list([X|Y]) :- write(X),nl, print_list([Y]).
```

Для описания этого предиката в разделе goal программы можно использовать конструкцию:

```
show :- dogs(X), print_list(X).
```

Если необходимо установить наличие некоторого элемента в списке, то применяется правило:

```
find_it( X,[X| _ ]).
find_it( X,[ _ |Y]):- find_it( X,[Y]).
```

В первом описании происходит сравнение объекта поиска и головы текущего списка. Если это сравнение неуспешно, то происходит откат и попытка повторения со вторым вариантом.

Программа 14

```
DOMAINS
    dog_list=symbol*
PREDICATES
    dogs(dog_list).
    find_it(symbol,dog_list).
GOAL
    find_it(«болонка»,[«лайка», «дог» ]),write(“да”).
CLAUSES
    find_it(X,[X,_]).
    find_it(X,[_,Y]):-find_it(X,[Y]).
```

Первое правило описывает ситуацию, когда искомый элемент X совпадает с головой списка.

Второе правило используется при неуспехе первого правила и описывает новый вызов первого правила, но уже с усеченным списком, в котором нет первого элемента и т.д. Если в списке нет элементов (пустой список), то второе правило оказывается неуспешным.

Программа не напечатает Yes**,** поскольку болонки нет в списке собак.

Следующий пример производит подсчет суммы всех элементов списка чисел.

Программа 15

```
DOMAINS
    spisok=integer*
PREDICATES
    summa_sp(spisok,integer)
```

```
CLAUSES
    summa_sp([],0).
    summa_sp([H|T],S):-summa_sp(T,S1),S= H +S1.
```

Рассмотрим операцию слияния двух списков.

Пусть L1 и L2 – две переменные, представляющие входные списки. L3 – выходной список, получаемый слиянием L1 и L2.

```
    append([ ],L,L).
    append( [N|L1], L2, [N|L3] ):- append(L1, L2, L3).
```

Первый вариант этого правила выполняется когда первый список пустой, второй работает в остальных случаях.

Рассмотрим использование этого предиката при следующем вызове:

```
    append([1,2,3], [4,5],_).
```

Здесь первый вариант правила сначала не работает, так как первый список не пустой.

Начинает работать второе правило, так как третий список пустой, к нему не применима операция деления на голову и хвост. Что касается первого списка, то он в процессе рекурсивных вызовов претерпевает деления на голову и хвост, и в конце концов становится пустым (его элементы попадают в стек)

```
    append([ ],[4,5],_).
```

После этого срабатывает первый вариант правила, и третий список инициализируется вторым:

```
    append([ ],[4,5], [4,5]).
```

И, наконец, чтобы завершить рекурсивные вызовы, ПРОЛОГ извлекает из стека (в обратном порядке) элементы первого списка и возвращает их, в соответствии с описанием второго варианта правила, в списки 1 и 3:

```
    append([1,2,3], [4,5],[1,2,3,4,5]).
```

Рассмотрим, как используются списки и механизм рекурсии при решении известной задачи о мужике, волке, козе и капусте.

Программа 16

```
DOMAINS
  LOC = east ; west                        /* описание берегов */
  STATE = state(LOC,LOC,LOC,LOC)  /* описание положения объектов */
  PATH = STATE*                            /* список состояний */

PREDICATES
  go(STATE,STATE)                          /* запуск алгоритма */
  path(STATE,STATE,PATH,PATH)  /* поиск пути к новому состоянию */
  move(STATE,STATE)                /* переход от состояния к состоянию */
  opposite(LOC,LOC)                /* описание противоположных сторон */
  unsafe(STATE)                            /* описание опасных состояний */
  member(STATE,PATH)               /* было ли такое состояние ? */
  write_path(PATH)                         /* вывод ответа */
  write_move(STATE,STATE)                  /* распечатка хода */
GOAL
        go(state(east,east,east,east),state(west,west,west,west)),
        write("solved press any key to continue"),
        readchar(_),
        exit.
CLAUSES
  go(S,G):-
        path(S,G,[S],L),
        nl,write("A solution is:"),nl,
        write_path(L),
        fail.
 go(_,_).
  path(S,G,L,L1):-       move(S,S1), not( unsafe(S1) ), not( member(S1,L) ),
                                   path( S1,G,[S1|L],L1),!.
  path(G,G,T,T):- !.  /* Конечное состояние найдено, список L копируется в L1 */
  move(state(X,X,G,C),state(Y,Y,G,C)):-opposite(X,Y).   /* мужик и волк */
 move(state(X,W,X,C),state(Y,W,Y,C)):-opposite(X,Y).   /* мужик и коза */
 move(state(X,W,G,X),state(Y,W,G,Y)):-opposite(X,Y).   /* мужик и капуста */
 move(state(X,W,G,C),state(Y,W,G,C)):-opposite(X,Y).    /* один мужик */

 opposite(east,west).
 opposite(west,east):-!.              /* описание противоположных сторон */
```

```
unsafe( state(F,X,X,_) ):- opposite(F,X)          /* волк съедает козу */
unsafe( state(F,_,X,X) ):- opposite(F,X).         /* коза съедает капусту */

member(X,[X|_]).
member(X,[_|L]):-member(X,L)     /* проверка состояний, которые уже были */

write_path( [H1,H2|T] ) :- !,
        readchar(_), write_move(H1,H2), write_path([H2|T]).
write_path( [ ] ).

        write_move( state(X,W,G,C), state(Y,W,G,C) ) :-!,
        write(“ Мужик пересекает реку с”,X,” на “,Y),nl.
        write_move( state(X,X,G,C), state(Y,Y,G,C) ) :-!,
        write(“ Мужик везет волка с “,X,” берега на “,Y),nl.
        write_move( state(X,W,X,C), state(Y,W,Y,C) ) :-!,
        write(“ Мужик везет козу с”,X,” берега на “,Y),nl.
        write_move( state(X,W,G,X), state(Y,W,G,Y) ) :-!,
        write(“ Мужик везет капусту с “,X,” берега на “,Y),nl.
```

8. РЕШЕНИЕ ЛОГИЧЕСКИХ ЗАДАЧ

Способности к описанию логических задач являются наиболее сильной стороной ПРОЛОГа.

Многие логические задачи связаны с рассмотрением нескольких конечных множеств с одинаковым количеством элементов, между которыми устанавливается взаимно-однозначное соответствие. На языке Пролог эти множества можно описывать как базы данных, а зависимости между объектами устанавливать с помощью правил.

Рассмотрим простую логическую задачу.

Пример:

«В велосипедных гонках три первых места заняли Алеша, Петя и Коля. Какое место занял каждый из них, если Петя занял не второе и не третье место, а Коля – не третье?»

Решение задачи заключается в установлении зависимости между спортсменами и местами, которую можно описать табл. 2 (прочерки указывают известную информацию):

Таблица 2

Имя	1-е место	2-е место	3-е место
Алеша			
Петя		–	–
Коля			–

Очевидно, Петя может занимать только первое место, тогда Коля – только второе, а Алеше достается третье (табл. 3).

Таблица 3

Имя	1-е место	2-е место	3-е место
Алеша	–	–	+
Петя	+	–	–
Коля	–	+	–

При описании этой задачи на ПРОЛОГе получается следующая программа.

Программа 17

```
PREDICATES
        name(symbol)
        mesto(symbol)
        prizer(symbol,symbol)
        solution(symbol,symbol,symbol,symbol,symbol,symbol)
CLAUSES
        name(alex). name(pier). name(nike).
        mesto(odin). mesto(dva). mesto(tri).
        prizer(X,Y):-name(X),mesto(Y),X=pier,not(Y=dva),not(Y=tri);
        name(X),mesto(Y),X=nike,not(Y=tri);
        name(X),mesto(Y),not(X=pier),not(X=nike).
        solution(X1,Y1,X2,Y2,X3,Y3):-name(X1),name(X2),name(X3),
                mesto(Y1),mesto(Y2),mesto(Y3),prizer(X1,Y1),
                prizer(X2,Y2), prizer(X3,Y3),Y1<>Y2,Y2<>Y3,Y1<>Y3,
                X1<>X2,X2<>X3,X1<>X3,!.
```

Конечно, в приведенном примере быстрее использовать таблицу для получения решения. Однако в более сложных случаях таблицы становятся многомерными. Рассмотрим еще один пример – логическую задачу (“Наука и жизнь” №3 , 1968):

«Пятеро студентов едут на велосипедах.

Их зовут Сергей, Борис, Леонид, Григорий и Виктор.

Велосипеды сделаны в пяти городах: Риге, Пензе, Львове, Харькове и Москве.

Каждый из студентов родился в одном из этих городов, но ни один из студентов не едет на велосипеде, сделанном на его родине.

Сергей едет на велосипеде, сделанном в Риге.

Борис родом из Риги, у него велосипед из Пензы.

У Виктора велосипед из Москвы.

У Григория велосипед из Харькова.

Виктор родом из Львова.

Уроженец Пензы едет на велосипеде, сделанном на родине Леонида.

Кто из студентов родом из Москвы ?»

Для решения этой проблемы можно предложить следующую программу:

Программа 18

```
 DOMAINS
        name=simbol
 PREDICATES
        student(name)          \* имя студента * \
        gorod(name)            \* название города *\
        velo(name,name)        \* владелец и «родина» велосипеда *\
        fact(name,name)        \* факты о принадлежности велосипедов *\
        fact1(name,name)       \* факты о месте рождения *\
        rodom(name,name)       \* описание места рождения студента *\
        rodom_penza(name)      \* описание для уроженца Пензы *\
CLAUSES
\* 1 *\     student(X):- X=serg; X=boris; X=vict; X=grig; X=leo.
\* 2 *\     gorod(Y):- Y=penza; Y=lvov; Y=moskva; Y=xarkov; Y=riga.
\* 3 *\     fact(serg,piga).
            fact(boris,penza).
            fact(vict,moskva).
            fact(grig,xarkov).
\* 4 *\     velo(X,Y):- student(X),gorod(Y), fact(X,Y), ! ;
            student(X),gorod(Y), not( fact(X, _ )),not(fact( _, Y)).
\* 5 *\     fact1(boris,riga). fact1(vict,lvov).
```

```
\* 6 *\     rodom_penza(X) :- student(X), Z=penza,not(fact1(X,_)),
           gorod(U),not(U=Z),velo(X,U),rodom(leo,U).
\* 7.1 *\   rodom(X,Z) :- student(X),gorod(Z),fact1(X,Z), ! ;
 \*7.2 *\   student(X),not(X=leo),Z=penza,rodom_penza(X), !;
\* 7.3 *\   student(X),gorod(Z),not(fact1(_,Z)),X=leo,not(Z=penza),
           student(K),not(fact1(K,_)),velo(K,Z);
 \*7.4 *\   student(X),not(X=leo),gorod(Z),not(Z=penza),not(fact1(_,Z)),
           not(fact1(X,_)), gorod(Y),not(Y=Z),velo(X,Y),
           not(rodom(leo,Z)),not(rodom(leo,Y)).
```

Рассмотрим описание фактов и правил в этой программе.

Первые два правила описывают возможные ограничения на значения предикатов student и gorod. Это необходимо, чтобы осуществлять допустимые подстановки при поиске решения.

Факты, обозначенные цифрой три, описывают известные нам данные о том, где сделаны велосипеды некоторых студентов.

Цифрой 4 обозначено правило, описывающее принадлежность некоторого велосипеда некоторому студенту. Правило состоит из двух альтернативных частей, разделенных знаком “;” . Первая часть правила говорит, что студент X владеет велосипедом Y, если мы знаем такой факт. Вторая часть правила позволяет делать любые подстановки, если это не противоречит известным фактам. Предикат отсечения “!” здесь прекращает поиск новых вариантов, если оказались выполненными все предшествующие ему условия. Эта операция называется отсечением.

Цифрой 5 обозначены известные факты о месте рождения студентов.

Цифрой 6 описаны представления о том, кто из студентов может быть родом из Пензы. Это не могут быть Борис или Виктор, поскольку мы знаем, что они родились в других городах, и должен быть студент, велосипед которого сделан на родине Леонида.

Состоящее из четырех частей правило 7 описывает общие представления о родине каждого из студентов. Во-первых, может быть известен такой факт и тогда другие варианты рассматривать нет необходимости. Вторая часть правила описывает, кто может быть родом из Пензы. Третья часть правила описывает возможное место рождения Леонида. И, наконец, четвертая часть правила описывает, из какого города, кроме Пензы, могут быть студенты, кроме Леонида. Загрузив эту программу, можно получить искомое решение.

Одной из часто встречающихся практических задач является задача составления расписаний. Рассмотрим пример подобной задачи (взятый из журнала “Наука и жизнь”):

«Пять студентов должны посещать лекции всю неделю, но по определенным ими установленным правилам, а именно:

1. Если пришли Андрей и Дмитрий, то Бориса быть не должно, но если Дмитрий не пришел, то Борис должен быть, а Виктор быть не должен.

2. Если Виктор пришел, то Андрея быть не должно и наоборот.

3. Если Дмитрий пришел, то Григория быть не должно.

4. Если Бориса нет, то Дмитрий должен быть, но если нет также и Виктора, а если Виктор есть, Дмитрия быть не должно, но должен быть Григорий.

5. Каждый день студенты должны приходить в разных сочетаниях. Какие это сочетания?»

Для решения этой проблемы может быть предложена программа:

Программа 19

```
DOMAINS
        s=symbol
PREDICATES
        st_A(s) st_D(s) st_B(s) st_V(s) st_G(s)
        ogr1(s,s,s,s,s,s)  ogr2(s,s,s,s,s,s)
        spisok(s,s,s,s,s)
        norm1(s,s,s,s,s,s)  norm2(s,s,s,s,s,s)
        norm3(s,s,s,s,s,s)  norm4(s,s,s,s,s,s)
CLAUSES
        st_A(A):-A=andre; A=net.
        st_D(D):-D=dmitri; D=net.
        st_B(B):-B=boris; B=net.
        st_V(V):-V=victor; V=net.
        st_G(G):-G=grig; G=net.
 ogr1(andre,_,_,net,_). ogr1(net,_,_,victor,_).
 ogr2(_,dmitri,_,_,net). ogr2(_,net,_,_,_).

 norm1(andre,dmitri,net,_,_).
 norm2(andre,net,boris,net,_).
```

```
norm3(_,dmitri,net,net,_).
norm4(_,net,net,victor,grig).

spisok(A,D,B,V,G):-st_A(A),st_D(D),st_B(B),st_V(V),st_G(G),
        norm1(A,D,B,V,G),ogr1(A,D,B,V,G),ogr2(A,D,B,V,G);
st_A(A),st_D(D),st_B(B),st_V(V),st_G(G),
        norm2(A,D,B,V,G),ogr1(A,D,B,V,G),ogr2(A,D,B,V,G);
st_A(A),st_D(D),st_B(B),st_V(V),st_G(G),
        norm3(A,D,B,V,G),ogr1(A,D,B,V,G),ogr2(A,D,B,V,G);
st_A(A),st_D(D),st_B(B),st_V(V),st_G(G),
        norm4(A,D,B,V,G),ogr1(A,D,B,V,G),ogr2(A,D,B,V,G).
st_A(A),st_D(D),st_B(B),st_V(V),st_G(G),
        not(norm1(A,D,B,V,G)),not(norm2(A,D,B,V,G)),
        not(norm3(A,D,B,V,G)),not(norm4(A,D,B,V,G)),
        ogr1(A,D,B,V,G),ogr2(A,D,B,V,G).
```

Возможно, может быть сформулировано более изящное программное решение, чем приведенное выше. При его поиске читатель может убедиться, что в подобных случаях очень большое значение имеет выбор предикатов и правил, которые могут определяться неединственным образом. При неудачном выборе правил можно получить заведомо ложные решения или исчерпать ресурсы компьютера при вложенном обращении одного правила к другому.

9. БАЗЫ ДАННЫХ И ЗНАНИЙ НА ПРОЛОГЕ

Факты, описанные в разделе clauses, можно рассматривать, как статическую базу данных (БД). Эти факты являются частью кода программы и не могут быть оперативно изменены. Для создания динамической базы данных в ПРОЛОГе предусмотрен специальный раздел database.

Предикаты в этом разделе могут иметь такую же форму представления, что и в статической части ПРОЛОГ-программы, но должны иметь другое имя.

Рассмотрим простой пример.

Программа 20

```
DOMAINS
        name = symbol
        rost, ves = integer
```

```
DATABASE
    dplayer(name, rost, ves)
PREDICATES
    player(name, rost, ves)
CLAUSES
    player("Михайлов", 180, 87)
    player("Петров", 187, 93)
    player("Харламов", 177, 80)
```

Допустим, что необходимо после запуска программы переместить данные из статической БД в динамическую. Для этого можно описать следующее правило:

```
assert_database:-player(N, R, V), assertz(dplayer(N, R, V)),fail.
```

В этом правиле использован встроенный предикат assertz, который помещает утверждение в базу данных после всех утверждений, которые там уже имеются. Есть также встроенные предикаты для удаления утверждений (retract), считывания с диска (consult), записи БД на диск (save) и сбора данных из БД в список (findall).

Главное достоинство БД на ПРОЛОГе, как и любой другой БД, заключается в возможности быстрого выборочного доступа к информации. Например, в приведенном выше примере это может быть запрос:

```
GOAL: player(N, R, V),R>180
```

Если сравнивать основные понятия ПРОЛОГа и реляционных БД, то получится следующая табл.4.

Таблица 4

ПРОЛОГ	Реляционные БД
Предикат	Отношение (таблица)
Объект	Атрибут отношения
Отдельное утверждение	Элемент отношения (запись)
Количество утверждений	Мощность отношения

Таким образом, можно сказать, что любая ПРОЛОГ-программа может быть названа базой данных.

Каково же главное отличие ПРОЛОГ-программ от реляционных БД? В БД можно только извлекать существующие сведения или изменять данные по заданному закону. В ПРОЛОГ-программе за счет использо-

вания правил и логического вывода можно получать новые знания. Поэтому ПРОЛОГ-программа может рассматриваться в качестве базы знаний (экспертной системы).

Рассмотрим пример простой экспертной системы, которая решает задачу определения вида экземпляра пойманной рыбы.

Программа 21

```
DATABASE
    xpositive(symbol, symbol)
    xnegative(symbol, symbol)
PREDICATES
    expertiza
    vopros(symbol, symbol)
    fish_is(symbol)
    positive(symbol, symbol)
    negative(symbol, symbol)
    remember(symbol, symbol, symbol)
    clear_facts
GOAL
    expertiza.
CLAUSES
    expertiza:-fish_is(X),!,nl,write("ваша рыба это ",X," "),nl, clear_facts.
    expertiza:-nl, write("это неизвестная рыба!"),clear_facts.
    vopros(X, Y):-write("вопрос – ",X, " ",Y, "? (да/нет)"),readln(R),
            remember(X,Y,R).
    positive(X,Y):- xpositive(X,Y),!.
    positive(X,Y):-not(negative(X,Y)),!,vopros(X,Y).
    negative(X,Y):-xnegative(X,Y),!.
    remember(X,Y,"да"):-assertz(xpositive(X,Y)).
    remember(X,Y,"нет"):-assertz(xnegative(X,Y)),fail.
    clear_facts:-retract(xpositive(_,_)),fail.
    clear_facts:-retract(xnegative(_,_)),fail.
    fish_is("сом"):- positive("у рыбы","вес >40 кг").
    fish_is("сом"):- positive("у рыбы","вес <40 кг"),
            positive("у рыбы","есть усы").
    fish_is("щука"):- positive("у рыбы","вес <20 кг"),
            positive("у рыбы","длинное узкое тело").
    fish_is("окунь"):- positive("у рыбы","вес <20 кг"),
```

```
        positive(“у рыбы”,”широкое тело”),
        positive(“у рыбы”,”темные полосы”).
fish_is(“плотва”):- positive(“у рыбы”,“вес <20 кг”),
        positive(“у рыбы”,”широкое тело”),
        positive(“у рыбы”,”серебристая чешуя”).
```

Нетрудно видеть, что *Программа 21* реализует заданное дерево поиска решения. Ответы на заданные вопросы позволяют продвигаться по ветвям этого дерева к одному из вариантов решения.

Задания для самостоятельной работы

1. Опишите двухместный предикат родитель и факты, относящиеся к этому предикату. Опишите предикаты для определения понятий отец, мать, дедушка, бабушка, дядя, тетя и т. п. С помощью запросов проверьте правильность полученного описания.

2. Напишите на ПРОЛОГе программу «Зоопарк», в которой описываются животные, их особенности, совместимость друг с другом и т.п.

3. Дана пара чисел – координаты точки на плоскости. Определите, попадает ли данная точка в круг единичного радиуса. Измените программу так, чтобы пользователь мог ввести координаты точки.

4. Найдите количество цифр во введенном числе.

5. Определите максимальную цифру введенного числа.

6. Определить максимальный элемент списка чисел.

7. Найти второй по величине элемент списка.

8. Сформировать новый список из тех элементов данного списка, которые стоят на нечетных позициях, например из списка чисел [1, 2, 3, 4, 5, 6, 7] получить [1, 3, 5, 7].

9. Трое ребят вышли гулять с собакой, кошкой и хомячком. Известно, что Петя не любит кошек и живет в одном подъезде с хозяйкой хомячка. Лена дружит с Таней, гуляющей с кошкой. Определить, с каким животным гулял каждый из детей?

10. Витя, Юра и Миша сидели на скамейке. В каком порядке они сидели, если известно, что Юра сидел слева от Миши и справа от Вити.

11. Пять пионеров Алик, Боря, Витя, Лена и Даша приехали в лагерь из 5 разных городов: Харькова, Умани, Полтавы, Славянска и Краматорска. Есть 4 высказывания:

1) Если Алик не из Умани, то Боря из Краматорска.

2) Или Боря, или Витя приехали из Харькова.

3) Если Витя не из Славянска, то Лена приехала из Харькова.

4) Или Даша приехала из Умани, или Лена из Краматорска.

Кто откуда приехал?

12. Четыре человека играют в домино.

Их фамилии Кузнецов, Токарев, Слесарев и Резчиков.

Профессия каждого игрока соответствует фамилии одногоиз других игроков.

Напротив Кузнецова сидит слесарь.

Напротив Резчикова сидит резчик.

Справа от Слесарева сидит токарь.

Кто сидит слева от кузнеца?

13. В одной школе уроки по истории, математике, биологии, географии, английскому и французскому языку вели три учителя – Морозов, Васильев и Токарев. Каждый из них преподавал два предмета.

Географ и учитель французского языка – соседи по дому.

Учитель биологии старше учителя математики. Морозов – самый молодой.

В понедельник первый урок по расписанию у Токарева, у биолога и у учителя французского языка.

В воскресенье Морозов, математик и учитель английского язык были на рыбалке.

Какие предметы преподает каждый учитель?

14. Есть четыре боксера: Томас Герберт, Герберт Френсис, Френсис Джеймс и Джеймс Томас.

Герберт намного сильнее Томаса.

Френсис сильнее и Томаса и Герберта.Герберт слабее Джеймса, но сильнее Френсиса.

В каком порядке нужно расположить боксеровот слабейшего к сильнейшему?

15. Имеется четыре котенка – Дружок, Елисей, Фантик и Мурлыка и четыре мальчика – Миша, Максим, Леня и Дима. Каждый мальчик взял себе котенка любимого цвета.

При этом:

1. Фантик – не рыжий

Мурлыка – не серый

2. Дружок – не белый

Елисей – не серый

3. У Миши – черный котенок

У Максима – Мурлыка
4. У Лени – Елисей
У Димы – белый котенок
5. Дима не взял Фантика
Дружок – не серый
Одно из этих пяти утверждений ложное

У какого мальчика какой котенок?

Вопросы для самопроверки

1. В чем отличие процедурных языков программирования от декларативных языков?
2. Что такое предикат и что такое местность предиката?
3. Когда является истинным одноместный предикат? Двуместный предикат?
4. Что такое атомарный предикат?
5. Как формулируется правило резолюции?
6. Что обозначают разделы domains, predicates, goal и clauses ПРОЛОГ-программы?
7. По каким правилам описываются переменные и константы в ПРОЛОГе?
8. Как обозначаются в ПРОЛОГ-программе основные логические операции И, ИЛИ, НЕ?
9. Как записывается логическое правило с несколькими посылками?
10. Какими способами могут выполняться запросы к ПРОЛОГ-программе?
11. Для чего используются анонимные переменные в запросах?
12. С какой целью используется предикат fail?
13. С какой целью используется предикат cut (!)?
14. Какие правила называются рекурсивными?
15. Как описывается момент окончания рекурсивных вызовов?
16. Какой структуре можно поставить в соответствие список?
17. С помощью какой операции описывается рекурсивная обработка списков в ПРОЛОГе?
18. Существует ли в ПРОЛОГе механизм для оперативного изменения набора фактов, с которыми работает программа?
19. В чем отличие ПРОЛОГ-программы от базы данных?
20. Какая структура базы данных соответствует одному предикату?
21. В чем заключаются основные ограничения при использовании ПРОЛОГа для описания объектов реального мира?

ЗАКЛЮЧЕНИЕ

Можно констатировать, что по большому счету ПРОЛОГ не занял в 90-е годы тех позиций, которые ему пророчили в начале 80-х. Нашумевший проект создания ЭВМ пятого поколения привел не к тем последствиям, которые от него ожидали. Здесь могут быть названы разные причины.

В практическом плане широкое внедрение графического интерфейса систем разработки программ и развитие визуального программирования придали «второе дыхание» традиционным процедурным языкам программирование. Широкую популярность завоевал объектно-ориентированный подход, позволяющий объединить описание объектов предметной области с процедурами их обработки, а также строить иерархические описания объектов с помощью механизма наследования.

В теоретическом плане самым важным является тот факт, что логика предикатов первого порядка, на которой построен ПРОЛОГ, имеет очевидные ограничения, поскольку требует структурно упрощать описываемую область. Говорить о доказательном выводе можно в сравнительно небольшом числе случаев, когда предметная область хорошо известна. Большинство же реальных задач относится к условиям неполноты и некорректности данных и знаний. Поэтому большинство существующих экспертных систем активно используют такие механизмы обработки неточных и неопределенных знаний, как нечеткая логика и нейронные сети.

Тем не менее, интерес к ПРОЛОГу не ослабевает, поскольку логическое мышление остается основным инструментом умственной деятельности человека. Возможно, эпоха массового использования ПРОЛОГа просто наступит несколько позже.

Библиографический список

1. *Симонс Дж.* ЭВМ пятого поколения: компьютеры 90-х годов. М.: Финансы и статистика, 1985. 174 с.

2. *Стерлинг Л., Шапиро Э.* Искусство программирования на языке Пролог: Пер. с англ. М.: Мир, 1990. 235 с.

3. *Ин Ц., Соломон Д.* Использование Турбо-Пролога: Пер. с англ. М.: Мир, 1993. 608 с.

4. *Янсон А.* Турбо-Пролог в сжатом изложении: Пер. с нем. М.: Мир, 1991. 95 с.

5. *Лорьер Ж.-Л.* Системы искусственного интеллекта. М.: Мир, 1991. 568 с.

6. *Уинстон П.* Искусственный интеллект. М.: Мир, 1980. 520 с.

7. *Хант Э.* Искусственный интеллект. М.: Мир, 1978. 558 с.

8. *Нильсон Н.* Принципы искусственного интеллекта. М.: Радио и связь, 1985. 376 с.

9. *Ковальски Р.* Логика в решении проблем. М.: Наука, 1990. 280 с.

10. Логический подход к искусственному интеллекту: от классической логики к логическому программированию: Пер. с фр. / *А. Тейз, П. Грибомон, Ж. Луи* и др. М.: Мир, 1990. 432 с.

11. Построение экспертных систем: Пер. с англ. / Под ред. *Ф. Хейеса-Рота, Д. Уотермана, Д. Ленната.* М.: Мир, 1987. 441 с.

12. *Уотермен Д.* Руководство по экспертным системам: Пер. с англ. М.: Мир, 1989. 388 с.

13. *Таунсенд К., Фохт Д*. Проектирование и программная реализация экспертных систем на персональных ЭВМ: Пер. с англ. М.: Финансы и статистика, 1990. 320 с.

14. *Попов Э. В.* Экспертные системы: Решение неформализованных задач в диалоге с ЭВМ. М.: Наука, 1987. 288 с.

15. *Элти Дж., Кумбс М*. Экспертные системы: концепции и примеры: Пер. с англ. М.: Финансы и статистика, 1987. 191 с.

16. *Бураков М. В., Попов О. С.* Интеллектуальные системы управления: Учеб. пособие/ ГААП. СПб., 1997. 108 с.

Содержание