МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

А.Г. КОКИН

ТУРБО ПРОЛОГ

Учебное пособие

Курган 2006

УДК 681.3.06:800.92
К60

Рецензенты: кафедра информатики и вычислительной техники Омского государственного педагогического университета (зав. кафедрой - д-р пед. наук, профессор З.В. Семенова; д-р пед. наук, профессор В.П. Пустобаев); д-р техн. наук, профессор кафедры информатики РГПУ им А.И. Герцена И.А. Румянцев.

Научный редактор: канд. физ. мат. наук, проф. В.А. Симахин

Печатается по решению методического совета Курганского государственного университета.

К60 Кокин А.Г. Турбо Пролог: Учебное пособие. – Курган: Изд-во Курганского гос. ун-та, 2006. - 94 с.

В учебном пособии излагаются вопросы, связанные с программированием на Турбо Прологе. Описываются приемы и средства организации программ, применение структур данных. Приведено большое количество примеров программ, иллюстрирующих возможности и особенности языка Турбо Пролог.

Учебное пособие предназначено для студентов, обучающихся по специальности 220400 “Программное обеспечение вычислительной техники и автоматизированных систем”, преподавателей, инженеров, а также специалистов, желающих самостоятельно изучить Турбо Пролог.

Рис.- 12, табл.- 6, библиограф. - 12 назв.

УДК 681.3.06:800.92

ISBN 5-86328-744-6

ВВЕДЕНИЕ

Турбо Пролог является декларативным языком, что означает использование дедукции при наличии определенных фактов и правил в логическом выводе при решении разнообразных задач программирования. Пользователь, работающий на Прологе, должен предоставить только описание задачи и основные правила ее решения, т.е. он должен определить цель, *что* следует найти. Пролог-система сама определит, *как* осуществить поиск решения – два аспекта Тубо Пролога: декларативный (что?) и процедурный (как?) [1, 2].

В этом коренное отличие языка Турбо Пролог от традиционных процедурных языков программирования.

Язык Турбо Пролог был создан усилиями многих ученых. Первая официальная версия Пролога была разработана в начале семидесятых годов прошлого века в Марсельском университете во Франции под руководством Алана Колмеруэра [3].

В настоящее время Турбо Пролог используется для программирования приложений, связанных с экспертными системами и искусственным интеллектом [4, 5].

Турбо Пролог – описательный язык. Программа на Турбо Прологе содержит *описание* данной задачи. Описание состоит из компонентов:

- имена и структуры *объектов* данной задачи;
- имена *отношений* между объектами (предикаты);
- *факты* и *правила*, характеризующие эти отношения.

Описание используется для определения отношений между заданными входными данными и выходными.

В Турбо Прологе используются факты, например, *сегодня идет дождь* и правила, например, *если сейчас дождь, а вы забыли зонтик, то вы промокните*.

Турбо Пролог умеет делать заключения. Пусть имеются факты:

Иван любит Машу.

Маша любит кошку.

и правило *Игорь любит X, если Маша любит X.*

Турбо Пролог сделает вывод, что *Игорь любит кошку*.

Факт – единичное утверждение: предикат(атом$_1$,...,атом $_n$).

Правило – итерационное утверждение вида A:-B$_1$,..., B$_n$, где

A, B - цели.

Приведенный выше пример запишется в виде:

likes(ivan, masha).
likes(masha, cat).
likes(igor, X):- likes(masha, X).

Предикат - функция, принимающая значение «истина» или «ложь». Предикат состоит из одного или нескольких предложений - утверждений, которые следуют друг за другом.

Как известно, Хорново предложение (утверждение) – это правило, факт или вопрос (запрос, цель). Цель - вопрос, записанный с использованием одного предиката [3].

Логическая программа - конечное множество правил и фактов (совокупность утверждений). Цель G логически следует из программы P если G может быть выведена из P с помощью конечного применения обобщенного правила modus ponens [4].

Резольвента - текущая цель, существующая на любой стадии вычислений. Значения логической программы P - множество выводимых из нее единичных целей.

Запрос - целевое утверждение, образуется и обрабатывается по тем же правилам, например, запрос для приведенного выше примера: кого любит Игорь?

Цель: likes(igor, X)

СИСТЕМА ТУРБО ПРОЛОГ

Турбо Пролог представляет типичный компилятор языка Пролог. С помощью его можно создавать программы для персональных компьютеров, использовать возможности аппаратуры, окна и полную цветовую графику. Любое окно может содержать как текстовую, так и графическую информацию.

Турбо Пролог обеспечивает доступ к памяти и портам ввода/вывода. Он представляет среду разработки интегрированных модульных программ. Модули, написанные на языке Пролог и других языках (таких, как С и язык Ассемблера), могут быть связаны между собой.

Стандартные предикаты для обработки файлов позволяют использовать файлы произвольного доступа.

Арифметика над целыми и вещественными числами встроена в систему и включает полный набор арифметических операций и функций.

Главное меню

После запуска системы Турбо Пролог на экране появляется главное меню системы Турбо Пролог и четыре системных окна.

Главное меню включает пункты: Files, Edit, Run, Compile, Options, Setup.

Edit

С помощью команды Edit осуществляется редактирование рабочего файла. Для знакомства с редактированием и запуском программ рассмотрим простую программу на Турбо Прологе Hello:

```
predicates
   hello
goal
   hello.
clauses
   hello:-
   makewindow(1,7,7,“Первая программа”,4,56,10, 22), nl,
    write(“Введите ваше имя”), nl,
    cursor(1,5),
    readln(Name), nl,
     write(“Привет”,Name).
```

Набирается программа в редакторе Edit. Выход из редактора (Esc) и запуск программы (Run). Турбо Пролог осуществляет проверку синтаксиса программы последовательно, строка за строкой. Если вы в разделе goal (цель) после предиката hello не поставили точку, то появится сообщение: 402 Syntax error, ‘.’ or ‘,’ expected in clause body (синтаксическая ошибка в теле предложения). Если неправильно набрали предикат, то появится сообщение: 404 Undeclared predicate or misspelling (необъявленный предикат или орфографическая ошибка). Необходимо исправить ошибки и запустить программу. В диалоговом окне появится Okno, в окне - Name. Введите свое имя, программа ответит:

Привет (ваше имя).

Для сохранения программы используется File - Save.

Для редактирования программы существует набор средств в меню Edit: F1 – Edit help. Здесь приводятся все системные предикаты (Show

help file), передвижения курсора (Cursor movement), вставка и удаление (Insert & Delete), функции блока с применением функциональных клавиш (Block functions), операции с блоком (Wordstar–like), разнообразные операции (Miscellaneous), глобальные функции (Global functions), горячие клавиши (Hot keys).

Команды редактора приведены в Приложении1.

Для осуществления поиска и замены отдельных строк написанной программы длиной до 25 символов используется клавиша F4. При нажатии F4 при редактировании текста программы появляется надпись Replace Text, Old text: (Заменить текст, старый текст:). Введите строку, подлежащую замене. При нажатии снова на F4 и появлении New text: введите новую строку. И снова F4. Появляется Global/Local replace (g/l):. Выберите опцию. При глобальном поиске будут участвовать все вхождения заменяемой строки, при локальном поиске – только очередное вхождение. Prompt before replacing (y/n): (Вы хотите, чтобы вас спрашивали, стоит ли замену делать). Отвечайте – нет (n).

Run

Созданная с помощью редактора программа компилируется и выполняется. Если программа содержит внутренние цели в разделе goal, то результат выводится в окне диалога.

Если цели являются внешними, т.е. отсутствует раздел программы goal, то задание целей происходит в окне диалога и там же выводится результат.

При выполнении программы некоторые функциональные клавиши имеют специальное назначение:

F8 – автоматически воспроизводит предыдущую цель;

F9 – вызывает редактор;

Э-F10 – изменяет размер или перемещает окно диалога;

Ctrl-P – вывод на печатающее устройство.

Compile

Созданная программа компилируется. Процесс и результаты компиляции зависят от выбранных параметров. Указывается три пункта компиляции:

- Memory (скомпилированная программа помещается в оперативную память);

- OBJ file (скомпилированная программа имеет формат объектного файла);

- EXE file (auto link) (скомпилированная программа имеет формат загрузочного файла. Установление связей автоматическое).

Пункт меню Project (all modules) используется для указания модулей в модульном программировании.

Link only – связывание модулей.

Files

Этот пункт содержит команды загрузки файла (Load), нового файла (New file), сохранения файла (Save), задания имени (Write to), выбора каталога (Directory),изменения каталога (Change dir), вызова MS-DOS с возвратом (OS shell), выхода (Quit).

Options

Link options (связи опций), Edit PRJ file (редактирование файла проекта в модульном программировании), Compiler directives (директивы компилятора)

Setup

При помощи этого пункта меню можно изменять параметры.

Colors – определение цвета фона и символов экрана;

Window size – изменение размера и положения окон;

Directories - определение оглавления;

Miscellaneous – сообщения об ошибках, размер стека;

Load SYS file – загрузка файла SYS с параметрами;

Save SYS file – сохранение значений параметров в файле SYS.

Системные окна

Системные окна: окно редактора - Editor, окно диалога - Dialog, окно сообщений - Message, окно трассировки - Trace. Этими окнами можно пользоваться в любой конфигурации: изменять их размер, перемещать. Для настройки окон используется меню Setup - Window size с применением клавиш <^> v.

Для временной настройки окон используется клавиша F6-Switch (переключить).

Editor

Системное окно Editor применяется для набора и редактирования программ. Возможно использование вспомогательного окна редактора Aux edit при нажатии клавиши F8.

Message

Окно сообщений последовательно воспроизводит все операции, со-

вершаемые с программой, и выводит сообщения об ошибках.

Dialog

Диалоговое окно служит средством обмена информацией с системой и реализует так называемый режим внешней цели. Пользователь может определить цель, что следует найти при решении данной задачи с помощью этого окна.

Trace

Трассировка позволяет проследить все последовательно выполняемые предикаты программы. Обратимся к программе Hello. Чтобы осуществить трассировку при выполнении программы, необходимо в начало программы добавить директиву компилятора trace. Запустите программу. Курсор в окне редактора указывает на цель goal, а в окне трассировки начало выполнения этой цели выглядит так: CALL: goal( ).

Для запуска пошагового режима выполнения программы надо нажимать клавишу F10. Нажав один раз, получим: курсор перешел на предикат hello, в окне трассировки добавилось

CALL: hello( ).

И так далее. Процесс завершится, когда в окне трассировки появится

RETURN: hello( )

RETURN: goal( )

Управление памятью в Турбо Прологе

Доступная память Турбо Пролога показана на рисунке 1.

→ 1	2 ←	3 ←	4	5

Рис. 1

1 – стек (Stack), 2 – неупорядоченный массив данных (Heap), 3 – след (Trail), 4 – код (Code), 5 – исходный текст (Source).

Стек используется для передачи параметров в рекурсивных программах. Чтобы допустить большее количество рекурсий в программах, размер стека следует увеличить. Размер окна может быть изменен с помощью опций команды Setup с заданием числа параграфов (1 параграф равен 16 байт) от 600 до 4000. Конфигурацию необходимо сохранить.

Неупорядоченный массив данных – область, используемая для построения структур, запоминания цепочек. Данная область используется также при вводе фактов в базу данных.

След – область, используемая для регистрации связывания и развязывания ссылочных переменных. По умолчанию пространство для следа не выделяется. Это можно изменить с помощью директивы компилятора trail.

Код – область, используемая для сгенерированного компилятором исходного текста. По умолчанию 16 Кбайт. Может быть изменено с использованием директивы компилятора code.

Исходный текст – область для исходного текста программы. Размеры исходного текста могут быть увеличены с помощью команды включения файла include.

Когда области стека, исходного текста, кода и следа установлены, оставшаяся память используется для неупорядоченного массива данных.

Стандартный предикат storage(StackSize, HeapSize, TrailSize)

возвращает размер трех областей памяти, используемых системой.

Директивы компилятора

Рядом свойств компилятора можно управлять с помощью директив компилятора, как ключевых слов в начале текста программы. Вот некоторые из них: check_determ, code, diagnostics, project, trace, trail (пункт меню Options).

Когда задается директива check_determ, предупреждение будет даваться для каждого программного предложения, которое приводит к неопределенным предикатам.

Директива code используется для задания размера области кода (по умолчанию – 16 Кбайт). Формат директивы: code = Number_of_paragraphs, где Number_of_paragraphs определяет число параграфов памяти (16 байт каждый), необходимое для области кода. Таким образом, запись code = 1024 устанавливает размер области кода 16 Кбайт.

С помощью директивы diagnostics компилятор отображает анализ программы. Выводится информация:

- имена используемых предикатов,
- все или не все предложения являются фактами,
- определенный предикат или нет,
- размер кода для каждого предиката,
- шаблон потока для каждого предиката,
- типы данных параметров.

Директива include используется в программах или модулях, когда содержимое файла должно быть включено в программу в течение компиляции. Формат директивы: include “dos_file_name”. Здесь dos_file_name имя файла операционной системы DOS.

Включаемые файлы могут быть использованы только в естественных границах программы, т.е. ключевое слово include может появиться только там, где допускается одно из ключевых слов domains, predicates, goal или clauses. На рисунке 2 показано включение файла в основную программу.

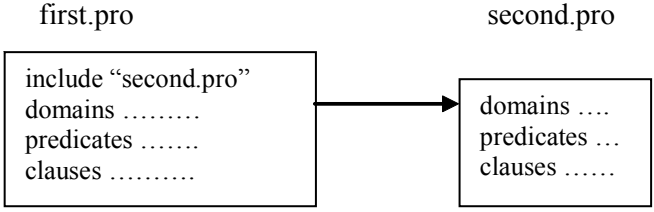


Рис. 2

Директива project используется в модульном программировании.

Директива trace показывает все возвраты (RETURN). Отслеживаются все предикаты или предикаты из списка, если он следует за директивой.

Сообщения из окна слежения: CALL – при вызове предиката в окне слежения отображается имя предиката и значения его параметров; RETURN – отображается в окне слежения, когда предложение выполнилось и предикат возвращает управление вызвавшему предикату; FAIL – отображается, когда предикат не достиг успеха; REDO – указывает на поиск с возвратом.

Директива trail используется для задания размера области слежения. Формат директивы: trail = число_слов

Область слежения используется для связывания ссылочных переменных. Если они используются, то необходимо задать размер области слежения. Обычно достаточно trail =100.

Эффективное программирование

Для сохранения пространства и ускорения выполнения программ Турбо Пролог устраняет остаточные рекурсии, где это возможно. Ите-

рационные операции могут быть реализованы рекурсивным образом с определенными требованиями по сопровождению рекурсий на стековом пространстве. Устранение остаточной рекурсии – это замена таких форм рекурсии на итерации. Там, где система сама не может устранить остаточную рекурсию, программист может ограничить ее действие применением нескольких правил о стиле программирования.

Первое правило - использовать большее количество переменных и меньшее количество предикатов.

Второе правило – записывать правила по принципу: сначала подцели с большим числом связанных переменных, затем остальные.

Третье правило - применять механизм унификации Турбо Пролога там, где это возможно. Например, проверку равенства двух списков с помощью предиката equal можно сделать следующим образом:

```
equal([],[]).
equal([Z|X],[Z|Y]):- equal(X,Y).
```

Гораздо проще: equal(X, X).

Четвертое правило - использовать поиск с возвратом (repeat…fail) и итерации вместо рекурсии для реализации повторений.

Рассмотрим замену рекурсии на итерации в примере вычисления факториала.

Рекурсивное вычисление факториала в Турбо Прологе:

```
% F равно факториалу N
factorial(N,F):-
 N>0, N1=N-1,
 factorial(N1,F1),
 F= N*F1.
factorial(0,1).        % – граничное условие
```

Итерационный цикл требует сохранения промежуточных результатов при приближении начальных значений к конечным результатам. В прологе отсутствуют локальные переменные для промежуточных результатов. Поэтому процедуры дополняются аргументами, называемыми накопителями. Накопители являются логическими переменными, а не ячейками памяти.

Итерационная программа на Турбо Прологе:

```
% F равно факториалу N
        factorial(N,F):- factorial(0,N,1,F).
 factorial(I,N,T,F):- I<N, I1=I+1, T1=T*I1,
```

```
factorial(I1,N,T1,F).
factorial(N,N,F,F).        % – граничное условие
```

Предикат factorial(I,N,T,F) является истинным, если F равно факториалу числа N, а значения переменных I и T совпадают со значениями соответствующих переменных в цикле перед (I+1) - итерацией цикла.

Счетчик цикла I. Процесс закончится, когда счетчик достигнет значения N. I = 0 – начальное значение счетчика, T1 используется для накопления текущего значения произведения, начальное значение T1=1.

Генерация исполнимых автономных программ

Программа на Турбо Прологе может быть скомпилирована и связана для формирования исполнимого файла. Турбо Пролог выполнит эту операцию при выборе пункта Compile, команды EXE file (auto link). При условии, что в программе нет ошибок, будет сгенерированы последовательно файлы OBJ, SYM, MAP, EXE с тем же именем.

Для того, чтобы гарантировать успех процесса связывания, должны быть выполнены следующие условия:

- папка с Турбо Прологом должна находиться в корневой папке диска C;
- редактор связей LINK.EXE должен присутствовать в справочнике OBJ;
- доступной памяти должно быть достаточно для связываемой программы.

Если имеется пакетный файл операционной системы DOS PLINK.BAT, то процесс связывания можно запустить вручную, для этого необходимо выйти из системы Турбо Пролог и вызвать редактор связей PLINK MYPROG. MYPROG – имя связанной программы.

При использовании системы Турбо Пролог PLINK получает автоматически три параметра:

- имя проекта или программы,
- текущий путь для справочника EXE,
- текущий путь для справочника TURBO.

При инициализации из операционной системы DOS должен быть задан, по крайней мере, первый параметр.

Модульное программирование

Система Турбо Пролог позволяет обрабатывать программы, раз-

битые на модули. Модули могут быть написаны, отредактированы и скомпилированы отдельно, а затем связаны вместе. Изменить и перекомпилировать можно только один из модулей при создании больших программ. По умолчанию имена всех предикатов и доменов являются локальными. Это означает, что в различных модулях можно использовать одни и те же имена.

Турбо Пролог использует две концепции для управления модульным программированием: декларации project и global.

Проект project

Турбо Пролог требует задания проекта project, определяющего имена модулей. Для этого файл LIBRARIAN должен содержать список модулей:

name_of_firstmodule+ (имя первого модуля)
name_of_secondmodule+ (имя второго модуля)

Каждый модуль определяется именем, за которым следует +. Файл проекта имеет расширение PRJ.

Первый шаг в модульном программировании состоит в выборе имени проекта и создании файла LIBRARIAN, содержащего имена модулей. Это выполняется с помощью опции Librarian в меню Setup. Каждый проект имеет свой LIBRARIAN.

Концепция проекта имеет две цели:

- содержимое файла LIBRARIAN используется в течение связывания, имена модулей вставляются в команду связывания;

- имя проекта используется в течение компиляции для идентификации символической таблицы для всех модулей проекта; для этого имя проекта должно быть задано в каждом модуле с помощью директивы компилятора project, которая имеет формат: project “MYPROG”.

Глобальные домены и предикаты

Программы Турбо Пролога взаимодействуют через границу модулей, используя предикаты, заданные в разделе global predicates.

Типы данных становятся глобальными посредством записи их в раздел global domains.

Декларации глобальных предикатов отличаются тем, что они должны содержать описания шаблонов потока и строятся по схеме: mypred(d1, d2,…,dn) – (f, f, …,f)(f, f,…,f)…, где d1, d2,…,dn – глобальные

домены, а каждая группа (f, f,…,f) обозначает шаблон потока (f – это либо i (входной параметр), либо o (выходной параметр)).

Компилирование и связывание модулей

Последовательность компилирования модулей:

1. Каждый модуль должен начинаться с двух директив компилятора, например:

```
project "myprog"
include "globals.pro"
```

Предполагается, что глобальные декларации сохранены в файле GLOBALS.PRO.

2. Только один модуль должен содержать раздел goal. Данный модуль считается главным модулем. Факты базы данных и данные должны быть записаны в главном модуле.

3. Модули в проекте могут быть скомпилированы либо на Compile – OBJ file, либо Compile – EXE file (auto link). Последний модуль необходимо компилировать на Compile – EXE file (auto link) . Процесс связывания выполняется автоматически.

Последовательность связывания модулей

Предположим, что проект называется MYPROG, два модуля проекта MAIN.PRO и SUB.PRO. Необходимые глобальные декларации находятся в файле GLOBALS.PRO.

Шаг 1. Создайте файл LIBRARIAN, задав имя MYPROG.PRJ и отредактировав его содержимое: main+ subi+. Сохраните файл MYPROG.PRJ.

Шаг 2. Создайте и сохраните файл глобальных деклараций GLOBALS.PRO:

```
global domains
    name=string
    global predicates
     welcome(name)-(i)
```

Шаг 3. Создайте файл главного модуля MAIN.PRO:

```
project «myprog»
include «globals.pro»
predicates
   test
goal
```

```
test.
clauses
  test:-clearwindow, write(«Введите ваше имя»),nl,
  readln(ThisName), welcome(ThisName).
```

Шаг 4. В меню Compile выполните команду OBJ file. Сгенерируются файлы MAIN.OBJ и MYPROG.SYM.

Шаг 5. Создайте и сохраните файл модуля SUB.PRO:

```
project «myprog»
include «globals.pro»
clauses
   welcome(Name):-write(«Welcome», Name),
   sound(300,200).
```

Шаг 6. В меню Compile выполните команду EXE file. Сгенерируется файл SUB.OBJ и выполнится процесс автосвязывания. В процессе связывания участвуют файлы MYPROG.SUM, MAIN.OBJ, SUB.OBJ, PROLOG.LIB, INIT.OBJ и получается файл MYPROG.EXE.

Взаимодействие процедур на разных языках

Турбо Пролог допускает взаимодействие с другими языками. Для того, чтобы информировать Турбо Пролог о том, что глобальный предикат реализован на другом языке, спецификация language добавляется к декларации глобального предиката:

```
global predicates
  add(integer, integer, integer)-(i, i, o), (i, i, i) language C
  scan(string, symbol)-(i, o) language Pascal
```

Турбо Пролог делает взаимодействующие языки явными для упрощения проблем активации формата записей и параметров, соглашений по вызовам и возвратам, , связывания и инициализации.

Процессоры предоставляют программисту два варианта вызова процедур: FAR и NEAR. Турбо Пролог требует, чтобы все вызовы процедур и возвраты были FAR. Для доступа ко всей памяти Турбо Пролог использует 32-разрядные указатели. Входные параметры помещаются непосредственно в стек.

Один и тот же предикат в Турбо Прологе может иметь различные варианты типов и различные потоки. Для вызова этих отдельных процедур должны быть различные имена. Например, в декларации

```
global predicates
```

add(integer, integer, integer)-(i, i, o), (i, i, i) language pascal

первый вариант с шаблоном потока (i, i, o) называется add_0, а второй с шаблоном потока (i, i, i) – add_I.

Приведенная ниже программа демонстрирует, как обращаться к подпрограмме, написанной на языке С, из программы на Турбо Прологе. Аналогично с подпрограммой, написанной на языке Паскаль.

Программа:

```
global predicates
    treble(integer, integer)-(i, o) language C
  goal
    write("Type in integer"), readint(f),
    treble(f, T),
    write("Treble that number is", T),nl.
```

Программа содержит обращение к treble, предложение language должно быть задано в разделе глобальных предикатов.

Графика в Турбо Прологе

Турбо Пролог предлагает выбор между графикой, основанной на точках и линиях, и полным набором команд "Графики с черепахой".

Графика на точках и линиях

Для работы в графическом режиме используется стандартный предикат graphics. Для возврата к текстовому режиму - стандартный предикат text. Предикат graphics имеет вид:

graphics(ModeParam, Palette, Background)

и выполняет начальные установки экрана со средней, высокой и сверхвысокой разрешающей способностью.

Возможные значения переменной ModeParam (параметр режима) и получающиеся в результате форматы экрана приведены в таблице 1.

Форматы экрана *Таблица 1*

Параметр режима	Число столбцов	Число строк	Описание
1	320	200	Средняя степень разрешения, 4 цвета
2	640	200	Высокая степень разрешения, черно-белое
3	320	200	изображение
4	640	200	Средняя степень разрешения, 16 цветов
5	640	350	Высокая степень разрешения, 16 цветов
			Сверхвысокая степень разрешения, 13 цветов

Цвета определяются одной из двух палитр, выбираемых в зависимости от того, с каким значением связана переменная Palette (палитра) – с 0 или с 1 (таблица 2).

Цвета экрана *Таблица 2*

Палитра	Цвет1	Цвет2	Цвет3
0	Зеленый	Красный	Желтый
1	Циан	Лиловый	Белый

Переменная Background (фон) имеет целое значение, выбираемое из таблицы 1.

Основными предикатами, используемыми в графике, являются предикат dot (точка) и line (линия).

Вызов предиката dot(Row, Column, Color) приводит к размещению точки в месте, определяемом значениями Row (строка) и Column(столбец). Первые два параметра – это целые значения в диапазоне от 0 до 31999.

Предикат line(Row1,Col1,Row2,Col2,Color) определяет линию.

Типичная последовательность вызовов стандартных предикатов, используемых для графики, приведена в программе:

```
goal
  write("Before graphics"),
  readchar(_),
  graphics(1,1,0),
  line(0,0,10000,20000,2),
  write("Ordinary write during graphics mode"),
  readchar(_),
  text,
  write("Alter graphics").
```

Графика с черепахой

При включении режима графики в середине экрана появляется черепаха. Она стоит вертикально, к хвосту ее прикреплено "перо". Движение черепахи управляется стандартными предикатами, "перо" оставляет след на экране.

Применение этих предикатов зависит от:

- расположение черепахи,

- направление движения,
- рисует “перо” или нет (активизировано ли оно),
- цвет пера.

Стандартный предикат pendown(перо вниз) активизирует перо, а предикат penup (перо вверх) приводит его в пассивное состояние. После вызова предиката graphics перо активизировано. Цвет следа определяется параметром предиката pencolor.

Движение черепахи управляется четырьмя стандартными предикатами: forward (вперед), back (назад), right (вправо), left (влево). Например, предикат forward(Step) показывает, на сколько шагов должна переместиться черепаха.

Чтобы повернуть черепаху, вводится переменная Angle (угол). Угол измеряется в градусах. Например, right(Angle) поворачивает черепаху вправо.

Некоторые графические объекты:

Треугольник

```
goal
  graphics(2,1,0),
  pendown,
  forward(5000), right(120),
  forward(5000), right(120),
  forward(5000), right(120).
```

Звезда

```
goal
  graphics(2,1,0),
  forward(5000),right(144),forward(5000),right(144),
  forward(5000),right(144),forward(5000),right(144),
  forward(5000),right(144),forward(5000).
```

Окружность

```
predicates
      circle1
  goal
     graphics(2, 1, 0),
     circle1.
  clauses
     circle1:-forward(100), right(4), circle1.
```

ОРГАНИЗАЦИЯ И УПРАВЛЕНИЕ ВЫПОЛНЕНИЕМ ПРОГРАММЫ

Структура, основные разделы программы

Раздел domains

Раздел domains содержит декларации доменов (типов данных). Турбо Пролог может оперировать с шестью стандартными типами данных (таблица 3). Объекты, принадлежащие литерным и строковым типам данных, могут содержать специальные символы:

\Число (символ со значением "Число" в кодировке ASCII);

\n (символ новой строки); \t (символ табулятора).

Если в декларации предикатов используются только стандартные типы данных, то в раздел domains в программе необязателен.

Типы данных *Таблица 3*

Тип данных	Описание	Внутримашинное представление	Пределы значений
Знаковый	char	ASCII, 1 байт	256 знаков
Целочисленный	integer	Целое, 2 байта	-32766 – +32767
Действительный	real	Число с плав. дес. точкой, 8 байт	1e-307 – 1e+308
Строковый	string	Последовательность литер	Конст. до 250 зн. Перемен. до 64К
Символьный	symbol	Последовательность символов или литер	До 250 знаков
Файловый	file	Файл	

Например, в приведенной ниже программе используются стандартные типы данных, раздел domains опущен. Программа запрашивает имя, по которому в базе данных отыскивается номер телефона.

```
Программа Телефон:
predicates
   reference(symbol, symbol)   % справка
goal
    write("Введите имя абонента:"),
    readln(Name),
     reference(Name, Phone),
    write("Телефон абонента", Name, Phone), nl.
```

```
clauses
  reference("Иван", "45-62-37").
  reference("Степан", "67-98-23").
 reference("Лена", "31-42-78").
 reference("Маша", "67-45-12").
```

После запуска программа просит ввести имя абонента. На введенное имя Лена Турбо Пролог выведет:

Телефон абонента Лена 31-42-78

При декларации типов данных в разделе domains используется четыре формата.

Первый формат: name = standard

Данная декларация объявляет домен name, состоящий из элементов стандартного типа standard, который может быть либо целым, либо действительным, либо знаковым, либо строковым, либо символическим. Например: apples, person = integer

Второй формат декларирует списковую структуру:

namelist= standard*

Эта запись декларирует тип данных list (список), например, для целочисленного списка: integerlist = integer* (* показывает, что в списке 0 или более элементов).

Третий формат декларирует сложную структуру:

имя_типа = имя _структуры(тип$_1$, тип$_2$,…, тип$_n$),

где имя_типа – тип порождаемой структуры, имя_структуры – функтор – имя композиции исходных типов, тип$_1$, тип$_2$,…, тип$_n$ – имена типов, каждый из которых может быть базовым или структурным. Имя_типа отождествляется с именем_структуры. Все имена с маленькой буквы.

Правила композиции типов: слева может быть указано не одно, а несколько разделенных запятыми имен нового типа; справа может быть указана не одна, а несколько структур, разделенных точками с запятой; при n=0 образуется 0-арная структура, задаваемая самим функтором.

Например:

```
data = data(day, month, year) % дата=дата(день, месяц, год)
day, year = integer
name, month = string
```

Четвертый формат декларирует файловую структуру:

file = name$_1$; name$_{2;}$ …,name$_N$

Программа может иметь только один домен данного типа file. Например, file = sales; buying (продажа, покупка).

Раздел predicates

Раздел служит для объявления несистемных предикатов, указываются их имена и типы каждого аргумента:

predname(tip_1, tip_2,…, tip_N),

где predname – имя предиката, а tip_1, tip_2,…, tip_N – определенные пользователем стандартные типы.

Рассмотрим программу Автомобиль:

```
domains
        brand, color = symbol   % марка, цвет
        age, price = integer    % возраст, цена
        mileage = real          % пройденные мили
predicates
   car(brand, mileage, age, color, price)
clauses
   car(renault, 20000, 3, red, 12000).
   car(ford, 90000, 4, gray, 25000).
   car(lada, 50000, 5, red, 8000).
```

Зададим вопрос: существует ли в базе данных машина стоимостью менее 25000 долларов? Для этого воспользуемся составной целью:

Цель: car(X, Y, Z, R, Cost), Cost<25000.

Турбо Пролог выдаст:

X= renault, Y=20000, Z=3, R=red, Cost=12000
X= lada, Y=50000, Z=5, R=red, Cost=8000

Необязательно назначать имена всем переменным в предикате car при запросах к базе данных. Для этого существует анонимная переменная, которая обозначается с помощью знака подчеркивания (_). Например, нас не интересуют марка, количество пройденных миль и цвет, но интересен возраст машины и цена, менее 25000. Тогда запрос будет выглядеть так: Цель: car(_, _, Age, _, Cost), Cost<25000.

Ткрбо Пролог выдаст:

Age = 3, Cost = 12000
Age = 5, Cost = 8000

Раздел clauses

Одним из основных разделов программы на Турбо Прологе являет-

ся раздел clauses (предложения), который содержит множество фактов и правил.

Рассмотрим следующий пример программы Увлечение:

```
domains
   person, vid_sporta =symbol
predicates
   likes(person, vid_sporta)
clauses
        likes(masha, tennis).          % Маша любит теннис
        likes(vladimir, football).     % Владимир любит футбол
        likes(ivan, swimming).         % Иван любит плавание
        likes(artur, tennis).          % Артур любит теннис
```

Раздел clauses приведенной программы полностью состоит из фактов. Каждый факт, заданный в разделе clauses, включает в себя отношение между объектами. Например, в факте likes(artur, tennis). отношением является likes, а объектами – artur и tennis. Имена объектов и отношений должны начинаться с маленькой буквы. В конце каждого факта ставится точка.

Если набрать программу и запустить ее, то в диалоговом окне появится сообщение: Цель:. Введем likes(X, tennis). Система выдаст:

X= masha
X= artur

Добавим в раздел clauses правило:

```
likes(maksim,X):- likes(ivan, X).
% Максим любит X если Иван любит X
```

На Цель: likes(maksim, swimming) система сопоставит правило с фактом likes(ivan, swimming) и ответит True.

В правиле likes(maksim,X):- likes(ivan, X). введена переменная X, обозначающая неизвестный вид деятельности. Имена переменных должны начинаться с заглавной буквы.

В диалоговом режиме (внешней цели) реализуется механизм возврата – поиск первого решения (доказательства истинности целевого предиката), отказ от него и поиск следующего решения в дереве целей. Этот процесс продолжается до тех пор, пока не будут получены все имеющиеся решения.

Рассмотрим программу Турнир, в которой содержатся факты - имена и возраст некоторых учеников класса, представляющую собой базу данных:

```
domains
   child = symbol
    age = integer
predicates
   pupil(child, age)   % ученик(ребенок, возраст)
clauses
pupil(petr, 9).
pupil(ivan, 10).
 pupil(stepan, 9).
 pupil(artur, 9).
```

Устроим турнир по пинг-понгу между всеми девятилетними учениками (по две игры на каждую пару). Воспользуемся целью:

pupil(Person1, 9), pupil(Person2, 9), Person1 <> Person2.

Цель означает, что необходимо найти Person1 возраста 9 лет и Person2 возраста 9 лет так, чтобы Person1 и Person2 были различны.

Первая подцель будет достигнута, если в качестве Person1 взять ученика petr. Если система возьмет в качестве второй подцели тоже petr, то третья подцель не удовлетворяется. Поэтому система возвращается к предыдущей подцели и берет в качестве Person2 ученика stepan. В этом случае все подцели удовлетворяются, цель достигнута. Пролог должен найти все возможные решения, поэтому система возвращается к предыдущей подцели и берет в качестве Person2 ученика artur. Найдено еще одно решение. В поисках последующих решений система возвращается ко второй подцели, но все возможности для этой подцели исчерпаны, поэтому осуществляется возврат к первой подцели. И так далее. В результате система выдаст все возможные решения:

```
Person1 =  petr      Person2 = stepan
Person1 =  petr      Person2 = artur
Person1 =  stepan  Person2 = petr
Person1 =  stepan  Person2 = artur
Person1 = artur      Person2 = petr
Person1 = artur      Person2 = stepan
```

Раздел goal

Программа может содержать раздел внутренней цели goal. Ввод информации и вывод результатов доказательства программной цели должен быть запрограммирован с помощью соответствующих системных предикатов.

Например, программа вывода кодов символов клавиатуры:

```
goal
        write ("Нажмите любой символ клавиатуры:"),
        readchar (X),
        char_int (X,Y), nl,
        write("ASCII–код введенного символа",X, "есть", У).
```

В конъюнктивной цели фигурируют системные предикаты: write(A,B,…) – вывод значений A, B,… в текущий выходной поток данных; readchar (X) – ввод одного знака из текущего входного потока данных с клавиатуры и связывание этого значения с именем X типа char; nl – переход на новую строку.

Приведем программу устройства свиданий. Софи ищет мужчину, который не курит или вегетарианец. Связка ИЛИ показывает, что необходимо ввести более, чем одно правило:

```
  sophie_could_date(X):-male(X),not(smoker(X)).
  sophie_could_date(X):-male(X), vegetarian(X).
```

Софи могла бы встретиться с X, если X мужчина и X не курит.

Софи могла бы встретиться с X, если X мужчина и X вегетарианец.

Программа Электронная сваха:

```
domains
   person = symbol
predicates
        male(person)                    % мужчина
        smoker(person)                  % курильщик
        vegetarian(person)              % вегетарианец
        sophie_could_date(person)
clauses
  male(stepan).
  male(vladimir).
  male(ivan).
  smoker(artur).
  smoker(ivan).
  vegetarian(stepan).
  vegetarian(ivan).
  sophie_could_date(X):-male(X),not(smoker(X)).
  sophie_could_date(X):-male(X), vegetarian(X).
goal
```

```
sophie_could_date(X), write("Возможный избранник
Софи есть", X), nl.
```

В результате выполнения программы получим результат:

```
Возможный избранник Софи есть stepan
```

Во-первых, в программе введено отрицание not(smoker(X)). Турбо Пролог будет считать это истиной, если невозможно доказать, что smoker(X) является истиной (т.е. что X – курильщик).

Во-вторых, присутствие цели внутри программы. Кроме того, появились системные предикаты write (вывод на экран), nl (новая строка). Перечень стандартных предикатов приведен в файле prolog.hlp или в Edit – F1 – Edit help.

Возможно введение комментариев строк символом %, комментариев блоков: начало - /*, конец - */.

Управление поиском решений

Турбо Пролог управляет поиском решений, соответствующих заданным целям.

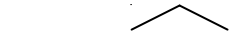
Рассмотрим поиск решений на примере программы Увлечения:

```
domains
   name, thing=symbol        % имя, дело = символ
predicates
   likes(name, thing)
    reads(name)
    inquisitive(name)        % пытливый(имя)
clauses
   likes(ivan, vine).
   likes(Z,books):- reads(Z), inquisitive(Z).
    likes(stepan(books).
    reads(ivan).
   inquisitive(ivan).
```

Зададим цель, состоящую из двух подцелей:

```
likes(X, vine), likes(X, books)
```

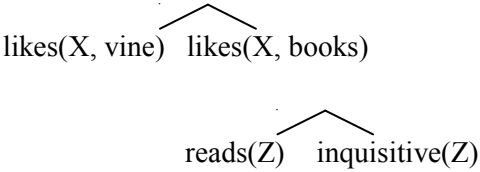
Во время обработки заданной цели Турбо пролог фиксирует, какие подцели были доказаны, какие – нет. Такой поиск может быть представлен в виде дерева целей:



Истинность подцелей доказывается слева направо. Следовательно, сначала доказывается подцель likes(X, vine). Турбо Пролог сопоставляет ее с первым предложением, переменная X связывается со значением ivan (X=ivan).

Затем Турбо Пролог пытается удовлетворить следующую подцель likes(X, books). Первое предложение likes(ivan, vine). не удовлетворяет подцель, Турбо Пролог проверяет следующее предложение: likes(Z,books):- reads(Z), inquisitive(Z).

Переменная Z соответствует X. Если подцель соответствует левой части правила, то необходимо рассмотреть его правую часть. Получаем дерево:



Дерево целей теперь включает подцели: reads(Z) и inquisitive(Z), причем переменная Z имеет значение ivan. Теперь Турбо Пролог будет искать факты, соответствующие обеим подцелям. Это reads(ivan). и inquisitive(ivan).

Первоначальная цель удовлетворена X=ivan.

Итак, если начало соответствующего предложения найдено и если существует другое предложение, которое может удовлетворить цель, то первое предложение помечается указателем, соответствующим точке возврата; все свободные переменные в подцели получают соответствующие значения (становятся связанными). Если соответствующее предложение представляет собой левую часть правила, то рассматривается и правая часть правила в качестве новой цели.

Организация возврата при поиске решений

В Турбо Прологе определен стандартный предикат fail (неудача) всегда ложный, который используется для организации возврата, отката. Чтобы продемонстрировать использование этого предиката, воспользуемся программой Отец:

```
domains
   name=symbol
predicates
```

```
    father(name, name)
    everybody                % каждый человек
clauses
    father(leonid, lena).
    father(petr, ivan).
    father(petr, stepan).
    everybody:- father(X,Y), write(X, " отец ", Y).
```

При запуске программы в режиме внешней цели: father(X,Y) Турбо Пролог выдаст:

```
X= leonid, Y= lena
X= petr, ivan
X= petr, stepan
```

Для цели: everybody Турбо Пролог выведет:

```
leonid  отец lena
```

Режим внешней цели обеспечивает автоматический вывод всех возможных решений (значений предметных переменных). В режиме внешней цели инициируется так называемый механизм отката – отказ от имеющегося и поиск нового отложенного решения, поэтому будут получены все решения.

Режим же внутренней цели не поддерживает ни того, ни другого. Для внутренней цели раздел goal:

```
goal
    father(X,Y),  write(X, " отец ", Y).
```

Турбо Пролог выведет только одно решение:

```
leonid  отец lena
```

Так как цель будет удовлетворена, истинность предиката father(X,Y) будет доказана.

Для организации перебора решений в базе данных для внутренней цели достаточно конъюнктивно завершить каждую альтернативную цель всегда ложным предикатом (fail). Этот ложный предикат, отрицая каждое очередное решение, будет вызывать откат, извлечет все возможные решения исходной цели и, к сожалению, объявит ее недоказуемой (ложной), что легко исправить, если исходную альтернативную цель дизъюнктивно завершить всегда истинной альтернативой true.

Внутренний раздел цели будет иметь вид:

```
goal
        father(X,Y),  write(X, " отец ", Y), nl, fail; true.
```

Турбо Пролог выведет:

leonid отец lena
petr отец ivan
petr отец stepan

Внутренний раздел цели для предиката everybody:

```
goal
  everybody, nl, fail; true.
```

Турбо Пролог выведет:

leonid отец lena
petr отец ivan
petr отец stepan

Отсечение предикатов

Иногда возникает необходимость отключать механизм возврата (отката) для сокращения пространства поиска. В Турбо Прологе имеется предикат, который при определенных обстоятельствах предотвращает возврат [5]. Такой предикат называется предикатом отсечения и обозначается в виде восклицательного знака ! .

Предикат отсечения всегда доказывается истинно с эффектом замораживания того решения, которое сформировалось к моменту доказательства предиката !. Отсечение совместно с откатом является средством осознанного сокращения пространства решений предиката.

После того, как система пройдет предикат ! , она больше не сможет возвратиться назад к подцелям, стоящим перед ним. Таким образом, предикат ! действует так, что система использует значения переменных, расположенных слева от предиката.

Рассмотрим пример:

```
a(X):- b(X),!,c(X), d(X).
b(e).
c(f).
d(g).
```

Без оператора ! на запрос a(Z) будет выдано: Z = e, f, g . С оператором ! программа выдаст единственный ответ Z = e.

После того, как предикат ! будет обработан, система не сможет вернуться назад для повторного рассмотрения цели b, если подцель c будет ложной при значениях переменной X.

Итак, отсечение отбрасывает все рассмотренные после него пред-

ложения, т.е. конъюнкция целей, стоящих перед отсечением, приводит к единственному решению.

Предикат отсечения имеет два основных применения:

а) когда использование предиката диктуется логикой программы;

б) когда заранее известно, что возвраты для поиска решения являются пустой тратой времени.

Красные отсечения

Отсечения, которые влияют на значение программы, называются красными [7]. Рассмотрим базу данных «возраст»:

```
возраст(иван, 19).
возраст(степан, 18).
возраст(игорь, 19).
```

Если первым аргументом предиката «возраст» будет константа (имя), а вторым – переменная (возраст), то запрос будет работать нормально.

Цель: возраст(иван, X),!. X=19

При данном запросе Турбо Пролог не будет искать значение еще одного возраста ивана.

Однако, если первым аргументом предиката «возраст» будет переменная, а вторым – константа, то система не найдет все правильные ответы на запрос:

Цель: возраст(X, 19),!. X=иван

Зеленые отсечения

Отсечение называется зеленым, если отбрасывает те пути вычисления, которые не приведут к новым решениям.

Обратимся к программе вычисления факториала:

```
factorial(1,1).
factorial(N,F):- N>1, N1=N-1, factorial(N1,F1), F= N*F1.
```

Условие N>1 необходимо, иначе второе предложение может быть удовлетворено при N=1. Без этого условия первый аргумент предиката factorial мог стать отрицательным, и программа вошла бы в бесконечный цикл.

Учитывая возможность использования предиката !, можно построить новые предложения:

```
factorial(1,1):-!.
```

```
factorial(N,F):- N1=N-1, factorial(N1,F1), F= N*F1.
```

Предикат отсечения показывает, что проверять второе предложение не следует.

Рассмотрим программу Общий интерес. Она основана на соображении, что два человека могут нравиться друг другу, если они имеют хотя бы один общий интерес.

```
domains
   name, interest = symbol
   interests = interest*
   sex= m; f
predicates
   findpairs                          % найденная пара
   person(name, sex, interests)
   member(interest, interests)
   common_interest(interests, interests, interest)
clauses
   findpairs:- person(Man, m, L1), person(Woman, f, L2),
               common_interest(L1, L2, _),
               write(Man, "  like ", Woman), nl, fail; true.
   common_interest(L1, L2, X):-
                          member(X, L1), member(X,L2),!.
   person(ivan, m, [computer, books, travel]).
   person(lena, f, [books, travel, swimming]).
   person(stepan, m, [volleyball,  boxing, rowing]).
   member(X,[X|_]).
   member(X,[_|T]):- member(X,T).
```

Интерес персонажей программы представляет собой списки интересов, которые декларируется следующим образом:

```
interests = interest*
```

Если хотя бы один интерес персонажей совпадает (для этого используется предикат member - принадлежность элемента списку и предикат !), то при цели: findpairs Турбо Пролог выведет сообщение:

```
ivan  like lena
```

При отсутствии отсечения получим избыточный результат:

```
ivan  like lena
ivan  like lena
```

Если отсечение поставить после предиката member:

```
member(X,[X|_]):-!.
```

то получается красное отсечение. В этом случае решения нет в связи с тем, что не происходит унификации первых элементов списков интересов персонажей.

Рекурсивные предикаты

Предикат menu

Своеобразие предиката menu заключается в том, что имя предиката фигурирует и в теле определения – рекурсивное определение:

menu:- menu.

В каждом цикле происходит откат и передоказательство уже доказанной части составного предиката. Отказ от отката и передоказательства осуществляется с помощью предиката !:

menu:-!, menu.

Отсечение необходимо, чтобы сказать системе – не пробуй остальные альтернативы, они все равно обречены на неудачу. С логической точки зрения определение предиката menu организует бесконечный процесс доказательства этого предиката: предикат menu истинен, если истинен предикат menu. Эта логическая бесконечность размыкается нелогическим способом – принудительным завершением программы предикатом exit.

```
Программа Меню:
predicates
   menu
clauses
   menu:- clearwindow,        % очистка текущего окна
   write("0 - Выход"), nl,
   write("1 - ……"), nl,
   write("2 - ……"), nl,   % пункты меню
          write("3 – ……"), nl,
   cursor(10,0),
   write( "Введите пункт меню:"),
   readint(N),                    % чтение целого числа
   clearwindow,
   go(N),                         %  предикат цели
   write("\n Нажмите ввод"),
   readchar(_),    % введен некоторый знак для  паузы
   menu.
```

```
goal
  makewindow(1,48,110,"Меню",2,15,20,50),
   menu.
```

Атрибуты предиката makewindow: 1- номер окна, 48 – атрибут окна, цвет поля – голубой, 110 – рамочный атрибут, цвет поля рамки – коричневый, "Меню" – заголовок окна, 2,15 – координаты левого верхнего угла окна, 20,50 – размер окна. В программе может быть несколько меню: menu1, menu2,…, они могут вкладываться одно в другое.

Продолжение программы, подцели меню:

```
predicates
    go(integer)
clauses
    go(0):- exit.
     go(1):- …….. .
     go(2):- …….. .
     go(3):- …….. .
```

Предикат repeat

Более радикальное решение одной и той же конъюнктивной цепочки предикатов осуществляется с помощью предиката repeat (rpt). Этот предикат генерирует бесконечное число циклов повторения поиска с возвратом. Он эквивалентен такому определению:

```
rpt.  rpt:-rpt.
```

Предикат repeat всегда истинен и имеет неограниченное число доказательств при откатах.

Поэтому,

```
rpt.
   rpt: -!, rpt.
```

Логическую бесконечность, содержащуюся в определении rpt:- rpt. можно разомкнуть логически обоснованным способом – добавлением к этому определению еще одной альтернативы, утверждающей безусловную истинность определяемого предиката rpt: rpt: - true; rpt.

Используя предикат rpt совместно с системным предикатом fail, легко описать любую многократно доказываемую конъюнктивную последовательность предикатов, поместив ее в предикатную пару rpt, …, fail. Например,

```
predicates
        rpt
clauses
        rpt.
         rpt:- !, rpt.
```

```
goal
    rpt, write("Введите имя"), readln(X), X=" ".
    rpt, write("Введите число"), readint(X), X=3.
```

Пока значение вводимой строки не станет «пусто» и значение числа равным 3, происходит повтор ввода значений, X=" " и X=3 заменяют в данном случае предикат fail.

Использование предиката rpt для организации меню:

```
goal
rpt,
        write("0 – Выход \ n",
              "1 – …….. \ n",
              "2 – …….. \ n ",
                ……),
        fail.
```

Окна Турбо Пролога

Задание атрибутов экрана

Турбо Пролог позволяет управлять такими характеристиками экрана как обратное изображение, подчеркивание и цвета. Эта информация передается стандартными предикатами с помощью значений атрибутов.

Для вычисления значения экранных атрибутов для цветовой графики (таблица 4) необходимо атрибуты фона и символов складывать. Сложение со 128 вызывает мерцание изображения.

Атрибуты окна *Таблица 4*

Фон	Символы
0 Черный	0 Черный
8 Серый	1 Синий
16 Синий	2 Зеленый
24 Голубой	3 Бирюзовый
32 Зеленый	4 Красный
40 Светло-зеленый	5 Лиловый
48 Бирюзовый	6 Коричневый
64 Красный	7 Белый
72 Светло-красный	
80 Лиловый	
88 Розовый	
96 Коричневый	
104 Желтый	
112 Белый	
120 Белый повышенной яркости	

Оконная система

Для создания окна используется предикат

makewindow(Номер, АтрЭкрана, АтрРамки, Заголовок,
Строка, Столбец, Высота, Ширина)

Определяется окно с заданными атрибутами и номером. Заголовок располагается над верхней рамкой. Параметры Строка и Столбец задают позицию левого верхнего угла окна.

Параметры Высота и Ширина определяют размер окна, они должны быть совместимы с размером экрана (25 строк по 80 символов). Курсор устанавливается в левый верхний угол окна.

Предикат clearwindow очищает текущее окно, предикат removewindow удаляет окно, активное в данный момент.

Предикат scroll(Число строк, Число столбцов) сдвигает содержимое текущего окна на заданное число строк и столбцов.

Предикат window_attr(Атрибут) определяет атрибуты окна.

Предикат window_str(Строка) записывает строку в текущее окно или считывает строку из текущего окна. Например, window_str("Hallo") - строка "Hallo" появляется в текущем окне; window_str(X) - содержимое окна связывается с переменной X.

Стандартный предикат cursor(Строка, Столбец) перемещает курсор в заданную позицию.

Предикат cursorform(НачСтрока, КонСтрока) устанавливает толщину и вертикальную координату курсора внутри области, занимаемой одним символом. Символ перекрывает 14 строк развертки, поэтому параметры могут принимать значения в диапазоне от 1 до 14.

Предикат field_attr(Строка, Столбец, Длина, ЗначАтр) устанавливает значение атрибута для заданного поля. Начало поля задается номерами строки и столбца. Например, field_attr(10, 20, 25, 104) - цвет фона на поле длины 25, которое начинается в позиции, определяемой строкой 10 и столбцом 20, становится желтым.

Предикат field_str(Строка,Столбец,Длина,СтрСимволов) записывает строку в поле, определяемое длиной и номерами строки и столбца.

Для системы окон используются предикаты:

gotowindow(НомОкна) и shiftwindow(НомОкна).

gotowindow(НомОкна) активизирует окно с заданным номером.

shiftwindow(НомОкна) меняет текущее окно или считывает номер текущего окна. Любое ранее активное окно сохраняется в предыдущем

состоянии. Например, shiftwindow(3) активирует окно с номером 3, shiftwindow(X) – X=1, если текущее окно имеет номер 1.

Средства обработки окон используются для построения игры в приведенной ниже программе: Отгадай слово.

Игрок должен отгадать одно за другим три слова по вводимым буквам. Если буква присутствует в слове, то она появляется в окне YES, в противном случае она попадает в окно NO. После каждой попытки угадать букву игрока просят отгадать все слово целиком. Для этого он должен набить его буква за буквой, нажимая каждый раз клавишу Enter.

Программа:

```
domains
  list=symbol*
   scores=integer
predicates
  member(symbol, list)
   run
  continue(list, scores)
  yes_no_count(symbol, list)
   quess_word(scores, list)
  word(list, integer)
  read_list(list, integer)
goal
  makewindow(1,30,30,» «,0,0,25,80),
  makewindow(2,46,30,»Counting»,1,20,4,36),
  makewindow(3,96,30,»YES»,5,5,7,30),
  makewindow(4,96,30,»NO»,5,40,7,30),
  makewindow(5,7,7,»»,14,20,10,34), run.
clauses
   run:-word(W,L),
  shiftwindow(1), clearwindow,
   write(«The word has «,L,»letters»),
  shiftwindow(2), clearwindow,
  shiftwindow(3), clearwindow,
  shiftwindow(4), continue(W,0),fail.
  continue(L,R):-shiftwindow(4), clearwindow,
   write(«Guess a letter: «),
  Total=R+1, readln(T), yes_no_count(T,L),
```

```
shiftwindow(4), clearwindow,
quess_word(Total,L), continue(L,Total).
yes_no_count(X,List):-
member(X,List), shiftwindow(2), write(X),!.
yes_no_count(X,_):-shiftwindow(3), write(X).
quess_word(Count,Word):-
 write(«Know the word yet? Press y or n»),
 readchar(A), A='y', cursor(0,0),
write(«Type it in one letter per line in»),
word(Word,L), read_list(G,L),
G = Word, clearwindow, window_attr(112),
write(«Right Tou used «, Count,»quess(es)»),
readchar(_), window_attr(7),!,fail.
 quess_word(_,_).
word([b, a, r, d], 4). word([p, r, o, l, o, g], L).
word([p, r, o, g, r,a, m], L).
 member(X,[X|_]).
member(X,[_|T]):-member(X,T).
read_list([],0):-!.
read_list([Ch|Rest],L):-
readln(Ch),L1=L-1, read_list(Rest,L1).
```

Окно в DOS

С помощью предиката system можно организовать доступ к системе DOS: system(Любая команда DOS или имя выполняемого файла).

Если аргумент представляет пустую строку, то система DOS будет вызвана при условии, что системный файл COMMAND.COM доступен из текущего каталога. Для возврата используется команда exit.

Средство вызова DOS можно сочетать со средствами обработки окон. Например, предикаты

```
makewindow(1, 7, 7, "DOS", 5, 20, 10, 40), system(" ").
```

ограничивают всякий диалог с DOS окном из 8 строк и 38 столбцов, расположенном в правом верхнем углу экрана.

Приведенная ниже программа представляет собой пример обслуживающей программы копирования файлов. В программе используется стандартный предикат concat (конкатенация).

Программа:

goal

```
makewindow(1, 7, 7,»Source»,0 ,0, 20, 35),
   write(«Whichfile doyou want to kopy?»),
cursor(3, 8), readln(X),
makewindow(2, 7, 7, «Destination», 0, 40, 20, 35),
 write(«What is the name of the new copy?»),
cursor(3, 8), readln(Y),
concat(X,» «,X1), concat(X1, Y, Z),
 concat(«copy «, Z, W),
makewindow(3, 7, 7, «Process», 14, 15, 8, 50),
write(«Copying «, X,» to «, Y), cursor(2, 7),
 system(W).
```

ОСНОВНЫЕ ЭЛЕМЕНТЫ ЯЗЫКА ТУРБО ПРОЛОГ

Имена, предложения

Имена используются для обозначения символических констант, доменов, предикатов и переменных. Имена состоят из букв или подстрочника (_), за которым следует любая комбинация букв, цифр [6].

Имена символических констант должны начинаться со строчных букв, имена переменных должны начинаться с заглавных букв.

Имена должны быть оригинальными, отражающими сущность именованного объекта. Наряду с нестандартными именами, задаваемыми пользователем, в Турбо Прологе имеются зарезервированные и особые имена, например, and, bound, random, database и другие.

Предложение – это либо факт, либо правило, соответствующее одному из декларированных предикатов. В общем случае предложение – это атом, либо атом с последующим двоеточием и тире, за которым идет список атомов, разделенных запятой или точкой с запятой.

Например, факт в Турбо Прологе отец(“Иван”, “Степан”). говорит о том, что Иван является отцом Степана.

Этот факт состоит из одного атома - отец и заключенного в скобки списка элементов – термов.

Терм – это константа, переменная или сложный элемент.

Термы

Простые константы

Простые константы относятся к одному из шести стандартных типов данных: знаковые относятся к типу char, целые относятся к типу integer, действительные числа относятся к типу real, строковые относятся к типу string, символические константы относятся к типу symbol, символическое имя файла относится к типу file. Все константы начинаются с маленькой буквы.

Переменные

Переменные – это имена, начинающиеся с заглавных букв, или подстрочник для представления анонимной переменной. Анонимная переменная используется, когда значение данной переменной не представляет интерес.

В Турбо Прологе различают два типа переменных:

- связанные переменные, унифицированные с термами, обладающие известными значениями;
- свободные переменные, не связанные с термами, о значении которых ничего не известно.

Предикат free(X) определяет, является ли данная переменная X свободной, предикат bound(X) проверяет, является ли переменная X связанной со значением.

Рассмотрим программу Хобби:

```
domains
   person, hobby = symbol
predicates
   likes(person, hobby)
clauses
   likes(lena, reading).
   likes(stepan, computer).
   likes(stepan, badminton).
   likes(leonid, badminton).
   likes(ivan, swimming ).
   likes(ivan, reading ).
```

Для этой программы зададим составную цель: кто любит читать и плавать?

Цель: likes(X, reading), likes(X, swimming).

Турбо Пролог проводит поиск слева направо и сверху вниз. В первой подцели система пытается связать свободную переменную X с определенным значением. Уже в первом факте likes(lena, reading). это ей удается. X принимает значение lena. Турбо Пролог устанавливает в базе данных указатель процедуры поиска.

Теперь надо удовлетворить вторую подцель. Турбо Пролог проверяет остальные факты, но вторая подцель оказывается ложной.

Турбо Пролог пытается найти другое решение первой подцели с учетом указателя, переменная X вновь считается свободной. Теперь переменная X для первой и второй подцели принимает значение ivan.

Составные термы

Турбо Пролог позволяет строить объекты, содержащие другие объекты. Такие объекты называются составными. Составные объекты - термы могут восприниматься и обрабатываться как единые объекты [7].

Составные термы состоят из описывающего имени – функтора и набора относящихся к нему других объектов (термов):

функтор(объект1, объект2,…).

При отсутствии термов функтор записывается как

функтор( ).

Например, составной терм включает функтор автор и три компоненты (объекта):

автор(имя, фамилия, год рождения).

Декларация составного терма автор:

```
domains
   author = author(firstname, lastname, year_of_birth)
   firstname, lastname = symbol
   year_of_birth = integer
```

Например, факт:

имеет(“Иван”, книга(“Дневной дозор”, “Лукьяненко”).

утверждает, что Иван имеет книгу “Дневной дозор”, которую написал Лукьяненко.

Для приведенного примера факта:

have(“Иван”, book(“Дневной дозор”, “Лукьяненко”).

декларация типов данных приводится в программе Библиотека:

```
domains
```

```
  book = book(title, author)
  title, author, name = symbol
predicates
   have(name, book)
clauses
  have("Иван", book("Дневной дозор", "Лукьяненко").
```

При цели: have("Иван", X) Турбо Пролог выдаст

X= book("Дневной дозор", "Лукьяненко")

Унификация термов

Одним из важных приемов Турбо Пролога является унификация (отождествление) и конкретизация переменных [2].

Процесс установления соответствия цели с предложением в Турбо Прологе называется унификацией.

Например, цель (целевое утверждение) собака(X) отождествляется с фактом собака(Рекс)., в результате чего переменная X конкретизируется и принимает значение Рекс: X=Рекс.

Рассмотрим программу:

```
domains
   title, author = symbol
    page = integer
  publication = book(title, page)
predicates
   written(author, publication)   % написана (автор, издание)
    novel(title)                  % роман(название)
clauses
    written("Фадеев", book("Разгром", 250)).
  written("Шолохов", book("Тихий Дон", 380)).
  novel(Title):- written(_, book(Title, Page)), Page>300.
```

Когда Турбо Пролог пытается удовлетворить указанную цель, он должен по очереди сопоставлять цель с предложениями предиката written в разделе clauses.

Например, для цели: written(X,Y), где X и Y – свободные переменные, имеется два решения:

X= Фадеев, Y= book("Разгром", 250)

X= Шолохов, Y= book("Тихий Дон", 380)

X связывается со значениями Фадеев и Шолохов, а Y – с

book(“Разгром”, 250) и book(“Тихий Дон”, 380).

Если задать цель: written(X, book(“Тихий Дон”, Y)), то будет выполняться унификация с первым предложением для предиката written. Свободная переменная X связывается со значением “Фадеев”, после чего делается попытка установить соответствие между book(“Тихий Дон”, Y) и book(“Разгром”, 250), что невозможно, так как константа “Тихий Дон” не унифицируется с константой “Разгром”.

Теперь Турбо Пролог пытается установить соответствие между written(X, book(“Тихий Дон”, Y)) и written(“Шолохов”, book(“Тихий Дон”, 380)). Это ему удается и он выдает:

X= Шолохов, Y=380

И, наконец, рассмотрим выполнение цели: novel(X)

Сначала достигается соответствие с левой частью правила:

```
novel(X)
   ↕
novel(Title):- written(_, book(Title, Page)), Page>300.
```

Далее Турбо Пролог делает первое предложение в правой части правила текущей подцелью, происходит унификация с первым фактом предиката written, при этом переменная Page связывается с константой 250. Теперь рассматривается второе предложение в правой части правила для предиката written: Page>300.

Оно становится текущей подцелью. Так как неравенство 250>300 является ложным, Турбо Пролог выбирает для доказательства истинности предиката written другое предложение раздела clauses и связывает переменную Title с константой “Тихий Дон”, а переменную Page – с константой 380, что приводит к решению:

X= Тихий Дон.

Алгоритм унификации в Турбо Прологе

Свободную переменную можно унифицировать с любым термом, в результате переменная связывается с этим термом.

Константа может быть унифицирована сама с собой или со свободной переменной.

Составной терм (структура) X можно унифицировать с другим составным термом Y, если оба они содержат один и тот же функтор и имеют одинаковое число аргументов. Кроме того, все термы должны быть унифицированы попарно.

Если X является константой или структурой, а Y –свободной переменной, то X и Y сопоставимы и могут быть унифицированы.

Таким образом, унификация обеспечивает присваивание значений переменным и доступ к структурам данных посредством общего механизма установки соответствия.

Предикаты

Системные (стандартные) предикаты

В состав системных предикатов входят часто употребляемые предикаты. Они известны системе, поэтому практическое освоение Пролога можно начинать с простых логических программ, содержащих один целевой системный предикат без теории. Все системные предикаты приведены в системном текстовом файле prolog.hlp.

Знакомство и изучение системных предикатов удобно начинать в диалоговом режиме (режим внешней цели). Для этого нужно войти в систему Турбо Пролог и организовать пустой программный файл с помощью пункта NewFile. При запуске (Run) доказательства пустого файла система открывает диалоговое окно и запрашивает (goal) внешний целевой предикат. Введем с клавиатуры один из системных предикатов, нажмем Enter и получим результат его доказательства. Для редактирования целевого предиката достаточно нажать F8.

Например, предикат char_int(A,B), предикатное имя char_int (character – знак, inter – целое). Смысл предиката char_int(A,B) – алфавитный знак A имеет внутримашинный целочисленный ASCII код B. Например, char_int('b', X) - знак b имеет ASCII код 98. 'b'–знаковая (литерная) константа. X - предметная переменная с типом integer. Истинность данного предиката составляет единственное значение X=98.

Составные предикаты

Составные предикаты строятся с помощью композиции предикатов (таблица 5). Могут быть получены выражения: (P, Q, R), (P, Q; R), (P; Q; R), (P:- Q; R) и т.п.

Правила композиции:

- отрицание not(p) может быть применено только к атомарному предикату p (нельзя not(not(p));
- целевое утверждение не может содержать импликации;
- не целевое составное предикатное выражение - правило может

содержать не более одной импликации Q:-P. Q – определяемый атомарный несистемный предикат (голова правила), атом, истинность которого определяется телом P. P – тело правила, определяющее условие истинности головы правила, – атом или составной предикат, не содержащий импликации.

Составные предикаты *Таблица 5*

Математическое название	Отрицание	Конъюнкция	Дизъюнкция	Импликация
Математическая нотация	$\neg$P	P$\wedge$Q или P&Q	P$\vee$Q	P$\rightarrow$Q
Нотация языка Турбо Пролог	not(P)	P and Q P,Q	P or Q P ; Q	Q if P Q :- P

- тело определения (правила) или целевое утверждение могут содержать лишь не сгруппированные скобками отрицания, конъюнкции и дизъюнкции атомарных предикатов (ДНФ). В записи составных предикатов запрещены группирующие скобки, связка по приоритетам;

- утверждением может быть только либо несистемный атомарный предикат–факт, либо определение (правило). В конце утверждения ставится точка.

Таким образом, в логической программе допустимы три вида утверждений: целевое (являющееся ДНФ атомарных предикатов), факт (факт–безусловно истинный, атомарный, несистемный предикат), правило–несистемный предикат, логически обусловленный комбинацией других предикатов и обобщающий смысл этой комбинации.

Отрицание предикатов

Множество истинности предиката not(char_int('b',X)) – любое число, отличное от 98. Такая комбинация запрещена, т.к. в Прологе отказались от генерации многоэлементных множеств истинности системных предикатов. Установлено жесткое ограничение – свободные переменные в предикате not запрещены [2].

Конъюнкция предикатов

Конъюнктивная цель истинна, если истинна каждая из ее подцелей. Доказательство составной цели проводится системой слева направо, в порядке следования подцелей.

Например, для целевой композиции X=3.14, Y=sin(ln(x))/exp(-x) реакция системы X=3.14, Y=21.0335. Результат доказательства первой подцели X используется при доказательстве второй подцели, результатом чего является значение переменной У.

Если эти же самые подцели соединить в противоположном порядке, то с логико-математической точки зрения этот составной предикат эквивалентен предыдущему (конъюнкция коммутативна), однако в силу установленного порядка доказательства первая подцель будет содержать недопустимую свободную переменную. Цель недоказуема.

Дизъюнкция предикатов

Дизъюнктивная цель представляет собой не одну, а несколько конъюнктивных целей, называемых альтернативными. Общая цель истинна, если истинна хотя бы одна из альтернативных целей. Альтернативные цели доказываются в порядке слева - направо или сверху - вниз.

Альтернативные цели в ЛП обладают свойством информационной независимости по предметным переменным. Это означает, что область локализации смысла любой из переменных ограничена той альтернативой, где находится данная переменная. Поэтому одноименные переменные в разных альтернативах имеют разный смысл. Например. Для целевой композиции X=1, Y=sin(X*X); Y=2, X=cos(2*X) реакция системы X=1, У=0.841; У=2, X=-0.6536.

При доказательстве цели X=1, Y=2; X=Y системой будет выдано предупреждение, что во второй альтернативе предикат X=У содержит две свободные переменные. После нажатия F10 система выдает множество истинности первой и второй альтернатив: X=1, У=2; X=_, У=_.

Импликация предикатов

Факт – это утверждение о существовании некоторого отношения между аргументами предиката. Правило – это факт, значение истинности которого зависит от истинности значений условий, образующих тело правила. Это высказывание имеет форму импликации: “если A то B”, где A – посылка, конъюнктивная цель, а следствие B – высказывание (унарный предикат). Определение предиката строится индуктивно в виде атомарного предиката B некоторой конъюнктивно – дизъюнктивной композиции A других определенных ранее предикатов [7]. Определение B:-A строится и читается справа - налево (если A, то B). Однако, опреде-

ление предиката может быть построено и дедуктивно: сначала формулируется следствие В, а затем отыскиваются условия А его истинности (В, если А). Импликация недопустима в цели.

Пусть имеется некоторая база данных по форме обучения студентов (дневная и вечерняя):

форма_обучения(иван, дневная).
форма_обучения(степан, вечерняя).
форма_обучения(марина, дневная).

Следующее правило устанавливает, что Иван знает Марину, если они вместе обучаются на дневной форме обучения:

знает(иван, марина):- форма_обучения(иван, дневная),
форма обучения(марина, дневная).

В нижеследующем запросе требуется выяснить, кого знает Иван. Цель: знает(иван, А)

А = марина

Успешность этого запроса зависит от успешности всех подцелей, входящих в тело правила.

Если вместо констант употребить переменные, то с помощью правила можно установить, что любые два человека знают друг друга, если они обучаются в одной форме обучения:

знает(А, В):- форма_обучения(А, С), форма_обучения(В, С).

Переменная Форма входит в обе подцели правила, в каждой подцели она должна иметь одно и то же значение.

Могут быть сформулированы следующие запросы:

Цель: знает(иван, В)
знает(степан, марина)
знает(А, степан)

Рекурсивные структуры данных

Организация рекурсии

Рекурсия обычно используется в двух случаях:

- когда отношения описываются с помощью самих отношений;
- когда составные объекты представляют собой часть других составных объектов (являются рекурсивными объектами) [6].

Рассмотрим первый случай на программе вычисления факториала.

Предикат factorial(N,F) имеет значение Истина, если F равно N!.

```
domains
        N, F= integer
predicates
   factorial(N, F)
clauses
   factorial(1,1).
   factorial(N,F):- N>1, N1=N-1, factorial(N1,F1), F= N*F1.
```

Проследим, каким образом проводится работа с факториалом при удовлетворении цели: factorial(2, F).

Воспользовавшись правилом, получаем:

factorial(2, Res):- 2>1, N1=2-1, factorial(N1,F1), F=2*F1.

Отсюда вытекает, что необходимо выполнить цель:

factorial(1, F1).

Учитывая факт factorial(1, 1)., эта цель удовлетворяется, если переменную F1 связать с 1. Теперь надо вычислить F=2*F1, что достигается, если связать переменную F с 2*1.

Рекурсивные структуры данных

Рекурсией можно воспользоваться при описании объектов, когда число элементов заранее неизвестно.

Рассмотрим задачу описания по имени всех учащихся в классе, если число учащихся заранее не известно.

Обозначим тип данных classlist (список класса) и начнем с описания пустого класса: classlist= empty (пустой)

Теперь напишем рекурсивное определение:

classlist=class(name, classlist).

Так, например, типичным объектом может быть: class(ivan, X),

что означает список класса, первым элементом которого является ivan, а X остальные учащиеся.

Класс из двух учащихся можно описать

class(ivan, class(lena, empty))

Отметим, что class(name, classlist) представляет собой составной объект, где class – функтор, name – соответствует одному учащемуся, а остальные учащиеся входят в список classlist.

Окончательная декларация состоит из двух альтернативных определений: classlist=class(name, classlist); empty.

В Турбо Прологе возможно рекурсивное объявление структурных типов – использование имени объявляемого типа в списке компонент определяющей его структуры. При этом, согласно методики рекурсивных определений, в объявлении типа данных должна быть указана хотя бы одна альтернатива, которая описывает предельно простую (не рекурсивную) структуру и является граничным состоянием рекурсии.

Рекурсивные объявления типа позволяют моделировать объекты высокой сложности и индуцируют адекватные им рекурсивные определения предикатов.

Линейные списки

Широко используемой рекурсивной структурой данных являются списки. Линейный список может быть определен как

list = l(element, list); end

где end – 0-арная структура – функтор – граничное условие, l – функтор – объединяет пару, первый член которой есть элемент некоторого определенного (не обязательно базового) типа, а второй – список. Он может быть либо пустым, завершающим рекурсию, либо непустым. Первый элемент списка – голова (head), а второй член списка – хвост (tail) [2].

Построим структуру – список из первых четырех чисел натурального ряда, считая, что головной элемент определен как целый тип:

element = integer

L= l(1,l(2,l(3,l(4)))).

Запишем предикат L= l(H,T), H=1 – голова, хвост - T=l(2,l(3,l(4, end))). Все имена задаются произвольно программистом.

Пример: программа определения суммы элементов списка:

```
  domains
    list=n; l(integer, list)
predicates
    listsum(list, integer)
goal
        L= l(1,l(2,l(3,l(4,n)))), listsum(L, Sum),
          write( “Сумма элементов списка”, L, “равна”, Sum), nl.
clauses
  listsum(n,0):-!.
  listsum(l(Head, Tail), Sum):- listsum(Tail, Sumtail),
                                        Sum=Sumtail + Head.
```

Сначала определяется сумма элементов хвоста списка, затем добавляется значение головы списка.

Запустив программу, получим: Сумма элементов списка равна 10

Деревья

Турбо Пролог позволяет строить составные объекты на нескольких уровнях. Например, в составном объекте

book(“Дневной дозор”, “Лукьяненко”)

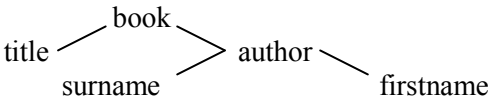
вместо фамилии автора можно ввести новую структуру, включающую также и имя автора. Если функтор нового составного объекта назвать словом author, то описание объекта – книги примет вид:

book(“Дневной дозор”, author(“Лукьяненко”, “Сергей”)).

Это приводит к следующим декларациям:

```
domains
   articles = (title, author)
   author = author(surname, firstname)
   title, surname, firstname = symbol
```

Сложный тип данных образует “дерево”:



Деревья представляют собой рекурсивную структуру [7]. Распространенной рекурсивной структурой являются бинарные деревья. Они определяются следующим образом:

btree = n; t(root, btree, btree),

где n – функтор пустого дерева (граничное состояние), root – корень дерева определенного, не обязательно базового, типа, btree, btree – два поддерева.

Рассмотрим приведенный выше пример:

T = t(book, t(title, n,n)), t(author, t(firstname, n,n), t(surname, n,n)).

Запишем предикат: T = t(Root, Left, Right),

```
Root = book,
Left = t(title, n,n)),
Right = t(author, t(firstname,  n,n), t(surname, n,n)).
```

Программа Обход дерева:

```
domains
   btree = n; t(string, btree, btree)
```

```
predicates
  writebtree(btree)          % вывод дерева
goal
  T = t(book, t(title, n,n)), t(author, t(firstname, n,n),
            t(surname, n,n)),  writebtree(T), nl.
clauses
  writebtree(n):-!.
  writebtree(t(Root, Left, Right)):- write(Root, " "),
  writebtree(Left), writebtree(Right).
```

Предикат writebtree реализует печать содержимого корней (узлов) бинарного дерева в порядке: корень – левое поддерево – правое поддерево – обход дерева слева – направо. Теоретически существует 6 вариантов обхода бинарного дерева, реализуемые перестановками трех предикатов печати - write(Root), writebtree(Left), writebtree(Right).

Обработка списков

Списки представляют собой основную структуру данных, используемую в программах на Турбо Прологе. Элементы списка разделяются запятыми и заключаются в квадратные скобки. Например, [1, 2, 5, 6], [dog, cat, canary].

Для описания домена для списков из целых чисел, можно воспользоваться такой декларацией:

```
domains
        integerlist = integer*
```

где * показывает, что в списке 0 или более элементов.

Объекты в списке могут быть другими списками, но все элементы списка должны принадлежать одному и тому же домену.

При построении списка необходимо наличие константного символа [ ] – пустой список для ограничения рекурсии.

Список обозначается [H|T], где H–голова списка, T–хвост списка ([1|2,3,4]). Унификация списков показана в таблице 6.

Приводятся некоторые операции по обработке списков.

Сумма элементов списка

Пусть необходимо вычислить сумму элементов списка. Для этого определим предикат, граничное условие выполнения операции и рекурсивное условие вычисления суммы.

Унификация списков *Таблица 6*

Список 1	Список 2	Связывание переменных
[X,Y,Z]	[Иван, любит, компьютер]	X = Иван, У = любит, Z= компьютер
[7]	[X|Y]	X=7, Y=[ ]
[1,2,3,4]	[X, Y| Z]	X = 1, Y = 2, Z = [3, 4]
[1, 2]	[3 | X]	Сравнения нет, головы списков различны

Предикат: listsum(L, Sum) -сумма элементов списка L равна Sum.

Граничное условие: listsum([ ],0):-!. (сумма элементов пустого списка равна 0).

Рекурсивное условие:

listsum([H|T], Sum):-listsum(T, SumT), Sum=SumT+H.

Сумма элементов списка равна сумме элементов хвоста списка (SumT) плюс элемент головы списка (H).

Программа:

```
domains
  integerlist=integer*
predicates
  listsum(integerlist, integer)
clauses
   listsum([ ],0):-!.
  listsum([H|T], Sum):-listsum(T, SumT), Sum=SumT+H.
```

Цель: listsum([3, -4, 5, 1, -7, 12], Sum)

Турбо Пролог выдаст: 10.

Пусть необходимо вычислить сумму положительных элементов списка. Так как обработка списка производится поэлементно и всегда начинается с головы, то раздел clauses будет выглядеть так:

```
listsum([],0):-!.
        listsum([H|T],Sum):-H>0,listsum(T, SumT), Sum=SumT+H;
                            listsum(T, SumT), Sum=SumT.
```

В этом случае в рекурсивном условии имеется два альтернативных определения, разделенных точкой с запятой (;). Первое - для положительных элементов списка, второе - для отрицательных.

Принадлежность элемента списку

Предикат: member(X,L). Отношение: X содержится в L, если X совпадает с головой списка, либо содержится в его хвосте:

XOL=[H|T] если X=H или XOT.

Рекурсивное условие:

- для головы списка: member(X, [X| _]:- !.
- для хвоста списка: member(X, [_ |Tail]):- member(X, Tail).

Первое предложение проверяет голову списка. Если голова соответствует переменной X, то X – член списка.

Если голова списка отличается от переменной X, то проверяется хвост списка по второму предложению.

Вывод элементов списка

Предикат writelist([H,|T]).

Граничное условие: writelist([ ]). – завершить выполнение, если элементов в списке больше нет.

Рекурсивное условие: writelist([H|T]):- write(H), nl, writelist(T). – вывод головы списка и хвоста списка.

Соединение двух списков

Предикат append(L1,L2, L3). Он объединяет списки L1, L2 в список L3. Если L1 пуст, то результат объединения L1 и L2 будет L2.

Граничное условие: append([ ], L, L).

Списки L1 и L2 объединяют в список L3, сделав голову списка L1 головой списка L3. Оставшаяся часть списка L3 получается за счет объединения хвоста списка L1 и всего списка L2. Если L1+L2 = L3, то [H|L1] + L2 = [H|L3].

Рекурсивное условие:

append([H|L1], L2, [H|L3]):- append(L1, L2, L3).

Индексирование списка

Предикат index([H|T],N,Y). N-й терм в списке есть Y. Задача получения N-терма в списке определяется следующим образом:

Граничное условие–первый терм в списке [H|T] есть H:

index([H|T], 1, H).

Рекурсивное условие – N-й терм в списке [H|T] является (N-1) термом в списке T: index([H|T],N,Y):- M=N-1, index(T,M,Y).

Обращение списка

Обращение списка с применением предиката append: рекурсивно

обращается хвост списка и затем добавляется первый элемент в конец обращенного хвоста.

Предикат reverse([H|T],Z). Z - обращенный список.

Граничное условие: reverse([ ],[ ]). – обращение пустого списка дает пустой список.

Рекурсивное условие:

reverse([H|T],Z):- reverse(T,X), append(X,[H], Z).

Длина списка

Предикат length(L,N) истинен, если длина списка L равна N элементов.

Граничное условие: length([ ],0). – длина пустого списка равна 0.

Рекурсивное условие: length([_|T],N):- length(T,Z), N=Z+1. – длина списка равна длине хвоста плюс голова списка.

Удаление элемента из списка

Предикат listdel(L1,X,L2) – список L2 получается из списка L1 удалением элемента X.

Граничное условие рекурсии: listdel([ ], X, [ ]). – удаление элемента X из пустого списка дает пустой список.

Две альтернативы рекурсивного определения:

- для головы списка - удаление X из списка [X|L1] приводит к L2, если удаление X из L1 приводит к L2:

listdel([X|L1),X,L2):- listdel(L1,X,L2).

- для хвоста списка:

listdel([H|L1],X,[H|L2]):- not(H=X), listdel(L1,X,L2).

Сортировка списков вставкой

Идея состоит в сортировке списка путем разделения его на две части, рекурсивной сортировке частей и последующем объединении частей для получения отсортированного списка, т.е. в рекурсивной перестановке хвоста списка и вставке головы списка в хвост так, чтобы упорядоченный хвост сохранил свою упорядоченность.

Предикат sort([H|T],Y) - список Y получается из отсортированного хвоста исходного списка T и вставленной в него головы исходного списка H.

Граничное условие: sort([ ], [ ]).

Рекурсивное условие: sort([H|T],Y):- sort(T,Z), insert(H,Z,Y).

Предикат insert(H,Z,Y) – элемент H вставляется в список Z и образуется список Y.

Граничное условие: insert(H, [ ], [H]).

Рекурсивное определение - если вставляемый элемент больше значения головы списка: insert(H,[H1|T1],[H1|Z]):- H>H1, insert(H.T1,Z).

меньше значения головы списка: insert(H,[H1|T1],[H,H1|T1]):- H<=H1.

Быстрая сортировка списков

Идея состоит в разделении списка путем выбора произвольного элемента списка и дальнейшего разбиения его на элементы, больше и меньше выбранного. Отсортированный список состоит из меньших элементов, за которыми следует выбранный элемент, и далее большие элементы. В качестве основы разбиения берется первый элемент. Используются два предиката quicksort([H|X1], Y1) и partition(X1, H, Littles, Bigs) (расчленение).

Рекурсивное определение:

```
quicksort([H|X1], Y1):- partition(X1, H, Littles, Bigs),
        quicksort(Littles, L1),    % сортировка меньших элементов
  quicksort(Bigs, B1),       % сортировка больших элементов
  append(L1, [H|B1], Y1).
  quicksort([ ],[ ]).          % граничное условие
```

Предикат partition(X1, Y, Littles, Bigs).

Если истинно выражение partition(X1, H, Littles, Bigs) и в голову списка добавлен элемент H<=Y, то истинным будет и

partition([H|X1], Y, [H|L1], B1).

Если добавить элемент H>Y, то истинным будет утверждение partition([H|X1], Y, L1[H|B1]).

Граничное условие: partition([ ],Y,[ ],[ ]).

РОДСТВЕННЫЕ ОТНОШЕНИЯ

Объектами данной области являются группы людей, связанные родственными узами [5,6]. Отношение родства есть утвердительное предложение вида: родство(X,У). Например, отец(X,У) – X является отцом

У. Аналогичным образом можно выразить остальные родственные отношения.

Виды родственных отношений

Отношение «родитель»

Для X и У - X является матерью или отцом У.

родитель(X,У):- мать(X,У).

родитель(X,У):- отец(X,У).

Отношение «отец»

Для X и У - X является отцом У, если X является родителем У и X – мужчина. отец(X, У):- родитель(X, У), мужчина(X).

Отношение «мать»

Для X и У - X является матерью У, если X является родителем У и X - женщина. мать(X,У):- родитель(X,У), женщина(X).

Отношение «сын»

Для X и У - X является сыном У, если У является родителем X и X – мужчина. сын(X,У):- мать(У,X), мужчина(X),

сын(X,У):- отец(У,X), мужчина(X),

сын(X,У):- родитель(У,X), мужчина(X).

Отношение «дочь»

Для X и У - X является дочерью У, если У является родителем X и X – женщина. дочь(X,У):- отец(У,X), женщина(X).

Отношение «сестра»

Для X и У - X является сестрой У, если у X и У есть общий родитель Z и X – женщина.

сестра(X,У):- родитель(Z,X), родитель(Z,Y), женщина(X).

Отношение «брат»

Для X и У - X является братом У, если у X и У есть общий родитель Z и X – мужчина.

брат(X,У):- родитель(Z,X), родитель(Z,Y), мужчина(X).

Отношение «дядя»

Для X и У – X является дядей У, если он является братом Z и Z является отцом У. ядя(X,Y):- брат(X,Z), отец(Z,Y).

Отношение «внук»

Для X и У - X является внуком У, если У является родителем Z и Z является родителем У и X – мужчина.

внук(X,У):- отец(У,Z), отец(Z,X), мужчина(X),

внук(X,У):- отец(У,Z), мать(Z,X), мужчина(X),
внук(X,У):- мать(У,Z), отец(Z,X), мужчина(X),
внук(X,У):- мать(У,Z), мать(Z,X), мужчина(X).

Отношение «дед»

Для X и У - X является дедом У, если X является отцом Z и Z является отцом У. дед(X,У):- отец(X,Z), отец(Z,У).

Отношение «предок»

Для X и У - X является предком У, если X является родителем У.
предок(X,У):- родитель(X,У),

Для X и У - X является предком У, если существует Z, такой что X – родитель Z и Z является предком У.
предок(X,У):- родитель(X,Z), предок(Z,Y).

Итак, отношение «предок»:
м предок(X,У):- родитель(X,У),
о предок(X,У):- родитель(X,Z), предок(Z,Y).

Вторая альтернатива утверждает, что если Z является предком У и X является родителем Z, то X также является предком У.

Можно ввести отношения вида:
son(X,Y,Z) - X является сыном матери У и отца Z
daughter(X,Y,Z) – X является дочерью матери У и отца Z
man(X,S,Y,Z) – человек X пола S имеет мать У и отца Z.
s = male; female %альтернативный тип данных

База данных о семейных отношениях

Рассмотрим генеалогическое дерево (рис.3):
Родственные отношения:
отец(алексей, степан). отец(степан, владимир).
отец(степан, света). отец(владимир, лена).
отец(владимир, иван). мать(нина, иван).
мать(нина, лена). мать(света, маша).
брат(иван, лена). сестра(лена, иван).
муж(владимир, нина). жена(нина, владимир).

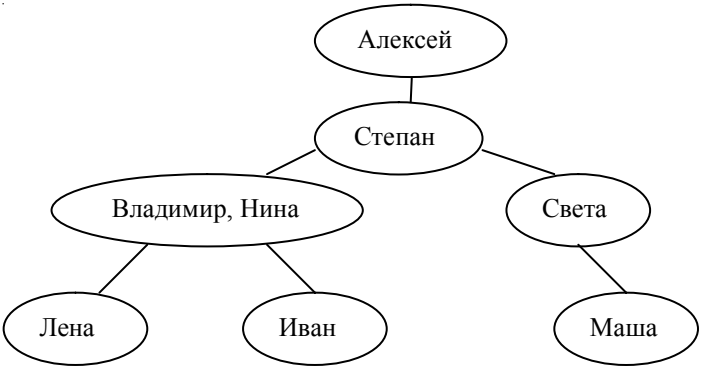


Рис. 3

Составим программу семейных отношений:

```
domains
      person = symbol
  predicates
      male(person)                    % мужчина(человек)
      female(person)                  % женщина(человек)
      father(person, person)          % отец(человек,человек)
      mother(person, person)          % мать(человек,человек)
       parent(person, person)          % предок(человек.человек)
       sister(person, person)          % сестра(человек,человек)
        brother(person, person)         % брат(человек,человек)
        uncle(person, person)           % дядя(человек,человек)
        grandfather(person, person)   % дед(человек,человек)
         ancestor(person, person)       % предок(человек,человек)
  clauses
     male(ivan). male(stepan). male(vladimir). male(aleksey).
      female(nina). female(masha). female(sveta). female(lena).
   mother(nina,lena). mother(nina,ivan). mother(sveta,masha).
   father(stepan,vladimir). father(vladimir,ivan).
    father(vladimir,lena). father(aleksey,stepan).
     father(stepan,sveta). parent(X,Y):-mother(X,Y).
   parent(X,Y):-father(X,Y). wife(nina,vladimir).
   husband(vladimir,nina).
```

```
brother(X,Y):-male(X),parent(Z,X),parent(Z,Y),X<>Y.
sister(X,Y):- female(X),parent(Z,X),parent(Z,Y),X<>Y.
uncle(X,Y):-mother(Z,Y),brother(Z,X).
uncle(X,Y):-father(Z,Y),brother(Z,X).
grandfather(X,Y):-father(X,Z),mother(Z,Y).
grandfather(X,Y):-father(X,Z),father(Z,Y).
ancestor(X,Y):-parent(X,Y).
ancestor(X,Y):-parent(X,Z), ancestor(Z,Y).
```

Теперь необходимо сформулировать соответствующие цели. Вот некоторые из них:

```
Родители и их дети
parent(X,Y)                % X является родителем Y
X=nina, Y=lena
X=nina, Y=ivan
X=sveta, Y=masha
X=ivan, Y=masha
X=vladimir, Y=lena
X=vladimir, Y=ivan
X=stepan, Y=sveta
X=aleksey, Y=stepan
Кто является братом Лены?
brother(X, lena)           % X является братом lena
X=ivan
Кто является дедом Ивана?
grandfather(X,ivan)        % X является дедом ivan
X=stepan
Кто является предком Маши?
ancestor(X,masha)          % X является предком masha
X=sveta
X=stepan
        X=aleksey
```

Интерес представляет программа, реализующая родственные отношения каждого члена семьи ко всем остальным. На каждом уровне иерархии для каждого члена семьи определяются родственные отношения ко всем членам семьи как по горизонтали влево и вправо, так и по вертикали вверх и вниз.

По горизонтали - это сестры и братья, двоюродные сестры и братья и т.д.

По вертикали вверх на следующем уровне иерархии - это отец, мать, дяди, тети, двоюродные дяди и тети и т.д.; далее на следующем уровне - это дедушки, бабушки, двоюродные дедушки, бабушки и т.д.

По вертикали вниз на следующем уровне иерархии - это сыновья, дочери, племянники, племянницы и т.д.; на следующем уровне - это внуки и внучки, двоюродные внуки и внучки и т.д.

БАЗА ДАННЫХ

Динамическая база данных

Турбо Пролог представляет реляционную базу данных как набор фактов, язык Турбо Пролог может быть использован как язык запросов для динамической базы данных (ДБД). Алгоритм унификации автоматически выбирает факты с соответствующими параметрами и конкретизирует значения для неизвестных параметров, а его алгоритм поиска с возвратом выдает все решения для данного запроса [8,9].

Динамическая база данных характеризуется возможностью ввода новых и удаления старых фактов в процессе выполнения программы.

Декларация базы данных следует за разделом domains и осуществляется с помощью ключевого слова database. Раздел database может иметь имя, например, database – db1. Программа может иметь несколько именованных разделов database, database без имени – безымянная ДБД.

Например, фрагмент базы данных Person:

```
domains
    name, address = string
    age = integer
    sex = male; female
database-bd1
    person(name, address, age, sex)
predicates
    male(name, address, age)
    female(name, address, age)
    child(name, age, sex)
clauses
    male(Name, Address, Age):- person(Name, Address,
                                      Age, Male).
```

person(“Иван”, “Москва”, 26, male).
person(“Степан”, “Томск”, 67, male).
person(“Нина”, “Курган”, 34, female).
person(“Лена”, “Курган”, 25, female).
……

При необходимости вывода всей базы данных в режиме диалога после запуска необходимо набрать цель:

person(X, Y, Z, R)

При организации вывода базы данных с использованием внутренней цели необходимо создать этот раздел:

```
goal
  person(X, Y, Z, R), writef(“%10%10%2%6\n”,X, Y, Z, R),
                                                nl, fail; true.
```

где writef – форматная строка, %10 – 10 позиций для X, Y, %2 – 2 позиции для Z, %6 – 6 позиций для R, \n –конец строки.

Предикат person может использоваться так же как и другие предикаты. Единственное отличие в возможности добавления и удаления фактов для предиката person в течение выполнения. Добавляемые факты запоминаются во внутренней памяти.

Манипуляции с базой данных реализуются с помощью стандартных предикатов.

Предикат assert(a,z)(Факт) вводит новые факты в начало (a) и в конец (z) записей базы данных.

Предикат retract(Факт) удаляет факты из базы данных.

Например, assertz(person(“Владимир”, “Москва”, 35,male)).

вводит факт как последнюю запись в базу данных для предиката person, а retract(person(“Степан”,_,_,_)). удаляет факт по имени “Степан”.

База данных может быть сохранена на диске в тестовом файле с помощью предиката save:

save(“имя файла DOS”, имя ДБД).

Для приведенного примера ДБД:

```
goal
    assertz(person(“Семен”, “Пермь”, 48, male)),
    save(“DBD1.dat”, bd1).
```

Файл DBD1.dat может быть считан в память с использованием предикат consult:

consult (“DBD1.dat”, bd1).

Предикаты findall и random

При создании базы данных наряду с другими используются предикаты findall и random.

Предикат findall (найти всех) используется, чтобы собрать в список значения, получаемые при использовании механизма возврата. Он имеет следующий вид:

findall(Variable, <atom>, ListVariable)

Первый параметр – переменная, порождающая значения, которые должны быть собраны в список. Второй параметр – это предикат который дает множество значений с помощью механизма возврата. Третий параметр – список значений для этой переменной.

В программе Возраст предикат findall используется для вывода среднего возраста нескольких человек.

```
domains
    name, address = string
      age = integer
    list = age*
predicates
    person(name, address, age)
   sumlist(list, age, integer)
goal
   findall(Age, person(_, _, Age), L), sumlist(L, Sum, W),
    Age = Sum/W, write("Average =  ",Age), nl.
clauses
   sumlist([], 0, 0).
   sumlist([H|T], S, W):-sumlist(T, S1, W1), S = H+S1, W = 1+W1.
    person("Иван", "Москва", 26).
    person("Степан", "Томск", 67).
   person("Нина", "Курган", 34).
    person("Лена", "Курган", 25).
```

Запустив программу, получим: Average = 38

Стандартный предикат random(X) возвращает вещественное число, удовлетворяющее неравенству 0<= X < =1.

В приведенной программе предикат random используется для случайной выборки трех имен из шести.

Программа Выбор:

```
predicates
```

```
    person(integer, symbol)
    rand_int(integer)
    rand_person(integer)
goal
    rand_person(3).
clauses
    person(1, ivan).
    person(2, stepan).
    person(3, oleg).
    person(4, artur).
    person(5, petr).
    person(6, vladimir).
    rand_int(X):-random(Y), X=Y*5+1.
    rand_person(0):-!.
    rand_person(Count):-
                rand_int(X), person(X, Name), write(Name), nl,
                NewCount=Count-1,rand_person(NewCount).
```

Запустив программу, получим:

ivan
oleg
petr

Обработка фактов

Предикат readterm делает возможным доступ к фактам базы данных в файле. Форма записи предиката:

readterm(name, TermParam).

где name – имя домена.

Рассмотрим использование предиката readterm с упрощенной базой данных bd2 в файле “GP.dat”:

```
p(“Иван”, “Москва”).
p(“Степан”,”Курган”).
p(“Алексей”, “Новгород”).
```

Фрагмент программы:

```
domains
    name, adr = string
    data_record = p(name, adr)
    file = file_data_record
```

```
predicates
    person(name, adr)
    boredata(file)
clauses
    person(Name, Adr):-openread(file_data_record,»GP.dat»),
                       readdevice(file_data_record),
                       boredata(file_data_record),
                       readterm(data_record, p(Name, Adr)).
    boredata(_).
   boredata(File):- not(eof(File)), boredata(File).
```

Запустим программу и определим цель: person(X,Y)

Турбо Пролог выдаст всю базу данных bd2:

X= Иван, Y= Москва
X= Степан, Y= Курган
X= Алексей, Y= Новгород

Если убрать предикат boredata, то Турбо Пролог выдаст только первую запись.

ЭКСПЕРТНЫЕ СИСТЕМЫ

Экспертная система (ЭС) – это программа, которая ведет себя подобно эксперту в некоторой предметной области. Она должна уметь принимать решения (советовать), иметь способность к объяснению своих решений [5,6,11]. Применение ЭС: проектирование логических схем, поиск неисправностей, диагностика, поиск компромиссных решений, принятие решений в критических ситуациях и т.д.

Обобщенная структура и функции ЭС

Обобщенная структура ЭС содержит три основных модуля:

- база знаний,
- система логического вывода,
- интерфейс с пользователем.

База знаний содержит знания, относящиеся к конкретной прикладной области - факты, правила, описывающие отношения или явления, а также методы и т.п.

Система логического вывода использует информацию базы знаний для получения решений.

Интерфейс с пользователем осуществляет обмен информацией между пользователем и системой.

Принято рассматривать систему вывода и интерфейс как один крупный модуль – оболочку ЭС.

ЭС должны решать задачи, требующие для своего решения экспертных знаний в некоторой конкретной области. ЭС должны обладать этими знаниями, поэтому их называют системами, основанными на знаниях. ЭС должна уметь объяснить свое поведение и свое решение пользователю так же, как это делает эксперт-человек. [3,4].

Функции ЭС:

- решение задач с использованием знаний о конкретной предметной области;

- взаимодействие с пользователем, включая объяснения во время и после окончания процесса решения.

Система логического вывода

Несмотря на все недостатки, наибольшее распространение получила продукционная модель представления знаний. При использовании продукционной модели база знаний состоит из набора правил. Программа, управляющая перебором правил, называется системой вывода.

Система вывода (интерпретатор правил) выполняет следующие функции: просмотр существующих фактов базы данных и правил из базы знаний, добавление новых фактов и определение порядка просмотра и применения правил.

Действие системы вывода основано на следующем: если известно, что истинно утверждение А и существует правило вида «ЕСЛИ А ТО В», тогда утверждение В также истинно - если истинна посылка, то должно быть истинно и заключение.

Система вывода, определяющая порядок выполнения правил, выполняет четыре функции:

- сопоставление – образец правила сопоставляется с имеющимися фактами;

- выбор – из нескольких правил выбирается одно наиболее подходящее по заданному критерию;

- действие – рабочая память подвергается изменению путем добавления в нее заключения найденного правила.

Система вывода работает циклически. В каждом цикле он про-

сматривает все правила, чтобы выявить те, посылки которых совпадают с известными на данный момент фактами из рабочей памяти. В этом случае заключение правила заносится в рабочую память и цикл повторяется. В одном цикле может быть найдено только одно правило.

Поиск правила происходит в прямом или в обратном направлении. При обратном порядке вывода вначале выдвигается некоторая гипотеза, а затем механизм вывода переходит к фактам, подтверждающим эту гипотезу. Если она оказалась правильной, то выбирается следующая гипотеза, детализирующая первую и являющаяся подцелью и т.п. Вывод такого типа называется управляемым целями. Применяется, когда цели известны и их немного.

В системах с прямым выводом по известным фактам отыскивается заключение. Прямой вывод называют выводом, управляемым данными. Циклический метод сочетает прямой и обратный метод.

Пример

Имеется фрагмент базы знаний из двух правил:

П1:Если «отдых летом» и «человек активный», то «ехать в горы».

П2: Если «любит солнце», то «отдых летом».

Поступили факты – «человек активный» и «любит солнце».

Прямой вывод: исходя из фактических данных, получить рекомендацию.

1-проход

Шаг 1 - П1 не работает – не хватает данных «отдых летом».

Шаг 2 - П2 работает, в базу поступает факт «отдых летом».

2-проход

Шаг 3 - П1 - активируется цель «ехать в горы» как совет ЭС.

Обратный вывод: подтвердить выбранную цель при помощи правил и данных.

1-проход

Шаг 1 - цель – «ехать в горы». Пробуем П1 – данных «отдых летом» нет, они становятся новой целью и ищется правило, в котором цель соответствует левой части.

Шаг 2 - цель - «отдых летом». Правило П2 подтверждает цель и активирует ее.

2-проход

Шаг 3 - П1 - подтверждается искомая цель.

Структурирование знаний

Поле знаний – это условное неформальное описание основных понятий (метаданных) и взаимосвязей между понятиями предметной области, выявленных из знаний эксперта, в виде графа, диаграммы, таблицы или текста [10].

Для формирования поля знаний и структурирования знаний можно выделить основные этапы:

1 Описание предметной области: определение понятий, явлений, процессов, предметов, действий, признаков.

2 Выявление связей между понятиями.

3 Выявление метапонятий и детализация понятий.

4 Построение пирамиды знаний. Под пирамидой знаний понимается иерархическая лестница понятий.

5 Определение отношений. При этом связям даются имена, а также обозначаются причинно-следственные, лингвистические, временные и другие виды отношений.

База знаний для идентификации животных

Описывается предметная область Животные, выявляются понятия (метаданные), связи и строится пирамида знаний, заканчивающаяся конкретными объектами – животными [5]. Фрагмент пирамиды знаний Животные показан на рис. 4.

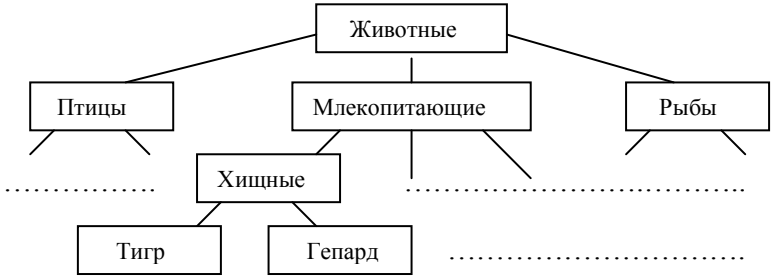


Рис. 4

Понятия – метаданные связываются между собой с помощью правил. При составлении правил нужно использовать минимальное доста-

точное множество условий при определении правил и избегать противоречия в правилах.

Некоторые из правил:

Правило 1: ЕСЛИ Животные имеют шерсть ИЛИ
Животные кормят детенышей молоком ТО
Животные эти - млекопитающие.

Правило 2: ЕСЛИ Животные имеют перья ИЛИ
Животные летают И
Животные откладывают яйца ТО
Животные эти - птицы.

……..

Правило 4: ЕСЛИ Животные эти - млекопитающие И
Животные едят мясо ТО
Животные эти - хищные.

……..

Правило 6: ЕСЛИ Животные это - хищные И
Животные имеют рыжевато-коричневый цвет И
Животные имеют черные полосы ТО
Животные эти - тигр.

……...

На Турбо Прологе правила будут иметь вид:

Животные эти млекопитающие:-
Животные имеют шерсть;
Животные кормят детенышей молоком.

Животные это хищнные:-
Животные эти млекопитающие,
Животные едят мясо.

……

Для идентификации животных внутри каждого подмножества нужно использовать список характеристик, признаков животных, количество которых будет определять точность идентификации.

Характеристики животных Тигра и Гепарда:

1 Имеют шерсть
2 Едят мясо
3 Имеют черные полосы
4 Имеют рыжевато-коричневый цвет
5 Имеют темные пятна

По набору характеристик идентифицируется животное:

Животные	Характеристики
Тигр	1, 2, 3, 4
Гепард	1, 2, 4, 5

В Турбо Прологе база знаний состоит из утверждений в форме rule (правило) и cond (условие). Предикаты rule содержат данные о животных, а предикат cond хранят условия (атрибуты), которые характеризуют различных животных. Фрагменты базы знаний (файл Mobile1.dbf):

```
rule(1,»Животные»,»Птицы»,[1],0)
rule(2,»Животные»,»Млекопитающие»,[2],0)
rule(3,»Животные»,»Рыбы»,[18],0)
.....
rule(14,»Млекопитающие»,»Хищные»,[12],1)
rule(17,»Хищные»,»Тигр»,[13,14],2)
rule(18,»Хищные»,»Гепард»,[13,15],2)
.....
cond(1,»Летают»)
cond(2,»Имеют волосы и кормят детенышей молоком»)
.....
cond(10,»Имеют темные полосы»)
cond(12,»Едят мясо»)
cond(13,»Имеют рыжевато-коричневый цвет»)
cond(14,»Имеют черные полосы»)
cond(15,»Имеют темные пятна»)
topic(«Животные»)
```

В каждом предложении rule первый аргумент – номер правила, второй – тип объекта и третий – объект. Список целых чисел [ ] задает номера условий из условных предложений cond, за ним следует номер уровня. Предложения cond содержат все характеристики животных.

Списки номеров условий служат для хранения множества фактов, в соответствии с которыми выбираются предложения rule.

Разработка оболочки

Для рассмотренной базы знаний приведен один из вариантов реализации программы. Чтобы программа могла идентифицировать некоторое животное (пусть это будет условно Кто-то-там), необходимо подобрать некоторые признаки Кто-то-там, имеющиеся в базе данных в виде

фактов. Для этого необходимо организовать диалог системы с пользователем так, как в следующем примере диалога (С – система, П – пользователь):

С: Это правда: Кто-то-там имеет шерсть?

П: Да.

С: Кто-то-там ест мясо?

П: Да.

С: Кто-то-там имеет рыжевато-коричневый цвет?

П: Да.

С: Кто-то-там имеет черные полосы?

П: Да.

С: Кто-то-там - это Тигр.

Кроме того, необходимо предусмотреть вход в базу знаний с любого уровня иерархии. Это необходимо как для идентификации животных, так и для расширения базы знаний - добавления новых уровней (категорий) и новых объектов. Например, для базы знаний Животные необходим вход с самого верхнего уровня Животные, с уровня Птицы, Млекопитающие, Рыбы и т.д.

Программа начинается с раздела domains:

```
code=2000
DOMAINS
  CONDITIONS = BNO*
  HISTORY = RNO*
  RNO, BNO, FNO = INTEGER
  CATEGORY = STRING
  file = save_file
  slist = string*
  key=integer
  LEVEL=integer
  y,z,x=integer
  S,SS=string
DATABASE    % Предикаты базы данных
  rule(RNO,CATEGORY,CATEGORY,CONDITIONS,LEVEL)
  cond(BNO,STRING)
  yes(BNO)  % Предикаты yes и no хранят ответы пользователя
  no(BNO)
  topic(string)
```

```
  lst(HISTORY)
database-temp
      s_list0(slist)
      s_list1(slist)
      s_list2(slist)
PREDICATES
  rpt
  menu(integer)
  menu2(integer)
  menu3
  key_menu(key,y)
  key_menu2(key,x,integer)
  key_menu3(key,y)
   key_menu4(key,y,integer,integer,integer,integer,integer,integer,CATEGORY)
  go(integer)
  go3(integer)
  chek
  add(integer,HISTORY,HISTORY)
  add_string(string,slist,slist)
      add_to_list(integer,CATEGORY)
     list_len(slist,integer)
  how(HISTORY)
  M_goal(CATEGORY)
  member(string,slist)
  unik(slist,slist)
  str_goal
  writelist(slist)
  run
  update
  setlevel(CATEGORY,integer)
  edit_rule
  list
  llist(HISTORY,string)
  load
  saves
  erase
  clear
```

```
    clear2
    clear3
    clear4
    clear_m
    erase_m
    listopt(integer,integer,integer,integer,integer,integer)
    term_in_list(slist,integer,string)
    evalans(char)
    reverse(CONDITIONS,CONDITIONS)
    reverse1(CONDITIONS,CONDITIONS,CONDITIONS)
    goes(HISTORY,CATEGORY)  % Предикаты механизма вывода
    check(RNO,HISTORY,CONDITIONS)
    notest(BNO)
    inpq(HISTORY,RNO,BNO,STRING)
    do_answer(HISTORY,RNO,STRING,BNO,INTEGER)
    sub_cat(CATEGORY,CATEGORY,CATEGORY)
    show_conditions(CONDITIONS,string)
    show_rule(RNO,string)
    show_cond(BNO,string)
    report(HISTORY,string)
    getrnr(RNO,RNO)
    getbnr(BNO,BNO)
    readcondl( CONDITIONS )
    getcond(BNO,STRING)
    save_y(char,string,string)
  CLAUSES
  /* Пользовательский интерфейс состоит из трех частей. Первая часть включает организацию меню, окон. Вторая часть включает механизм выбора и инициализацию процесса поиска и сопоставления по образцу. Третья часть задает и получает ответы от пользователя и взаимодействует с механизмом вывода.*/
    rpt. rpt:-!,rpt. chek:-rule(_,M,_,_,_),isname(M). add(X,L,[X|L]).
    add_string(X,L,[X|L]).
    add_to_list(0,Cat):-s_list0(L1),add_string(Cat,L1,L),retractall(s_list0(_)),
      assert(s_list0(L)).
    add_to_list(1,Cat):-s_list1(L1),add_string(Cat,L1,L),retractall(s_list1(_)),
       assert(s_list1(L)).
```

```
add_to_list(2,Cat):-s_list2(L1),add_string(Cat,L1,L),retractall(s_list2(_)),
        assert(s_list2(L)).
list_len([],0). list_len([_|T],Len):-list_len(T,Len1),Len=Len1+1.
        term_in_list([H|_],1,H). term_in_list([_|T],N,X):-M=N-
1,term_in_list(T,M,X).
load:-consult(«Mobile1.dbf»),assert(lst([])).
saves:-clear3,clear4,retract(lst(_)),save(«Mobile1.dbf»),assert(lst([])).
clear:-retract(yes(_)),retract(no(_)),fail;true.
erase:-retract(_),fail;true.
clear2:-retract(lst(_)),fail;true.
clear3:-retract(yes(_)),fail;true.
clear4:-retract(no(_)),fail;true.
clear_m:-retract(yes(_)),retract(no(_)),fail;shiftwindow(3),
write(«Очистка произведена\n»),menu(5).
erase_m:-retract(_),fail;shiftwindow(3),write(«Правила
выгружены.\n»),
menu(5).
str_goal:
-shiftwindow(3),rule(_,A,_,_,Level),add_to_list(Level,A),fail;true.
key_menu(27,_):-cursorform(6,8),exit.
key_menu(0,Y):-inkey(Key),key_menu(Key,Y).
key_menu(72,Y):-field_attr(Y,2,19,112),Y>2,!,Z=Y-
1,cursor(Z,0);cursor(9,0).
key_menu(80,Y):
-field_attr(Y,2,19,112),Y<9,!,Z=Y+1,cursor(Z,0);cursor(2,0).
key_menu(13,Y):-Y=9,cursorform(6,8),exit;go(Y).
menu(YY):-shiftwindow(1),clearwindow, nl,nl,
        write(« · Загрузка правил    «),nl,
        write(« · Запуск             «),nl,
        write(« · Сохранение         «),nl,
        write(« · Выгрузка правил    «),nl,
        write(« · Просмотр правил    «),nl,
        write(« · Добавление правил  «),nl,
        write(« · Редактирование     «),nl,
        write(« · Выход              «),
cursor(YY,0),!,rpt,cursor(Y,_),field_attr(Y,2,19,113),inkey(Key),
        key_menu(Key,Y),fail.
```

```
go(3):-not(chek),!,shiftwindow(3),
  write(« _____\n»),
  write(«|                              |\n»),
  write(«|База знаний не загружена|\n»),
  write(«|_____|»),nl,menu(3);run.
go(2):-not(not(chek)),!,shiftwindow(3),write(«База знаний уже загружена!\n»),
  menu(3); load,str_goal,shiftwindow(3),
  writef(«\nБаза знаний загружена»),nl,menu(3).%загрузка .
go(4):-not(chek),!,shiftwindow(3),
  write(« _____\n»),
  write(«|                              |\n»),
  write(«|База знаний не загружена|\n»),
  write(«|_____|»),nl,menu(4);
  saves,shiftwindow(3),write(«База знаний сохранена.\n»),menu(4).
go(5):-erase_m.
go(6):-not(chek),!,shiftwindow(3),
  write(« _____\n»),
  write(«|                              |\n»),
  write(«|База знаний не загружена|\n»),
  write(«|_____|»),nl,menu(6);list.
go(7):-update;cursorform(14,15),shiftwindow(2),clearwindow,clear3,clear4,
  retract(lst(_)),assert(lst([])),menu(7).
go(8):-not(chek),!,shiftwindow(3),
  write(« _____\n»),
  write(«|                              |\n»),
  write(«|База знаний не загружена|\n»),
  write(«|_____|»),nl,menu(8);edit_rule.
key_menu3(27,X):-clearwindow,erase,load,menu(2).
key_menu3(75,X):-field_attr(7,X,3,112),X>7,!,Z=X-6,cursor(7,Z),fail.
key_menu3(77,X):-field_attr(7,X,3,112),X<11,!,Z=X+6,cursor(7,Z),fail.
key_menu3(13,X):-go3(X).
menu3:-shiftwindow(2),clearwindow,cursor(4,0),
  write(« Хотите пополнить базу знаний? «),cursor(7,0),
  write(«  Да  Нет  «),cursor(7,7),rpt,cursor(_,X),
  field_attr(7,X,3,113),inkey(Key),key_menu3(Key,X),!.
```

```
go3(7):-update;cursorform(14,15),shiftwindow(2),clearwindow,clear3,
clear4,
  retract(lst(_)),assert(lst([])),menu(2).
go3(13):-saves,erase,load,clearwindow,menu(2).
getrnr(N,N):-not(rule(N,_,_,_,_)),!.
getrnr(N,N1):-H=N+1,getrnr(H,N1).
getbnr(N,N):-not(cond(N,_)),!.
getbnr(N,N1):-H=N+1,getbnr(H,N1).
getcond(BNO,COND):-cond(BNO,COND),!.
getcond(BNO,COND):-getbnr(1,BNO),assert(cond(BNO,COND)).
readcondl([BNO|R]):-write(«условие: «),readln(COND),COND><«»,!,
  getcond(BNO,COND),readcondl(R). readcondl([]).
update:-not(chek),!,shiftwindow(3),
  write(« _____\n»),
  write(«|                                |\n»),
  write(«|База знаний не загружена|\n»),
  write(«|_____|»),nl,menu(7);
  shiftwindow(4),clearwindow,cursorform(6,8),
cursor(3,3),write(«Категория: «), cursor(5,3),write(«Подкатегория:
«),cursor(3,20),readln(KAT1),KAT1><«»,
cursor(5,20),readln(SUB1),SUB1><«»,readcondl(CONDL),getrnr(1,RNO),
setlevel(KAT1,Level3),assert(rule(RNO,KAT1,SUB1,CONDL,Level3)),update.
setlevel(Cat,Lev):-rule(_,Cat,_,_,L),!,Lev=L;rule(_,_,Cat,_,L),Lev=L+1.
run:-M_goal(Mygoal),nl,nl,goes([],Mygoal),!.
run:-shiftwindow(3),cursorform(14,15),
  write(«Экспертная система не нашла ответа.\n»),menu3.
key_menu4(0,Y,B1,E1,B2,E2,B3,E3,_):-inkey(Key),key_menu4(Key,Y,
B1,E1,B2,E2,B3,E3,_).
key_menu4(72,Y,B1,_,_,_,_,B3,E3,_):-
field_attr(Y,0,78,112),Y=B1,!,cursor(E3,0).
key_menu4(72,Y,B1,E1,B2,_,_,_,_):-
field_attr(Y,0,78,112),Y=B2,!,cursor(E1,0).
key_menu4(72,Y,_,_,B2,E2,B3,_,_):-
field_attr(Y,0,78,112),Y=B3,!,cursor(E2,0).
key_menu4(72,Y,B1,E1,_,_,_,_,_):-field_attr(Y,0,78,112),
    Y>B1,Y<=E1,!,Z=Y-1,cursor(Z,0).
key_menu4(72,Y,_,_,B2,E2,_,_,_):-field_attr(Y,0,78,112),
```

```
        Y>B2,Y<=E2,!,Z=Y-1,cursor(Z,0).
    key_menu4(72,Y,_,_,_,_,B3,E3,_):-field_attr(Y,0,78,112),
        Y>B3,Y<=E3,!,Z=Y-1,cursor(Z,0).
    key_menu4(80,Y,_,E1,B2,_,_,_,_):-field_attr(Y,0,78,112),Y=E1,!,
cursor(B2,0).
    key_menu4(80,Y,_,_,_,E2,B3,_,_):-field_attr(Y,0,78,112),Y=E2,!,
cursor(B3,0).
    key_menu4(80,Y,B1,_,_,_,_,E3,_):-field_attr(Y,0,78,112),Y=E3,!,
cursor(B1,0).
    key_menu4(80,Y,B1,E1,_,_,_,_,_):-field_attr(Y,0,78,112),
        Y>=B1,Y<E1,!,Z=Y+1,cursor(Z,0).
    key_menu4(80,Y,_,_,B2,E2,_,_,_):-field_attr(Y,0,78,112),
        Y>=B2,Y<E2,!,Z=Y+1,cursor(Z,0).
    key_menu4(80,Y,_,_,_,_,B3,E3,_):-field_attr(Y,0,78,112),
        Y>=B3,Y<E3,!,Z=Y+1,cursor(Z,0).
    key_menu4(13,Y,_,_,_,_,B3,_,Gol):-cursor(V,_),V>=B3,
clearwindow,s_list2(L),
      unik(L,L1),Number=V-B3,Num=Number+1,term_in_list(L1,Num,Gol).
    key_menu4(13,Y,_,_,B2,_,_,_,Gol):-cursor(V,_),V>=B2
,clearwindow,s_list1(L),
      unik(L,L1),Number=V-B2,Num=Number+1,term_in_list(L1,Num,Gol).
    key_menu4(13,Y,B1,_,_,_,_,_,Gol):-cursor(V,_),V>=B1,
clearwindow,s_list0(L),
      unik(L,L1),Number=V-B1,Num=Number+1,term_in_list(L1,Num,Gol).
    M_goal(Mygoal):-shiftwindow(8),clearwindow,listopt
(B1,E1,B2,E2,B3,E3),
        cursor(B1,0),!,rpt,cursor(Y,_),field_attr(Y,0,78,113),inkey(Key),
        key_menu4(Key,Y,B1,E1,B2,E2,B3,E3,Mygoal),bound(Mygoal),!.
    writelist([]). writelist([H|T]):-write(«      «,H,»\n»),writelist(T).
    member(H,[H|_]). member(H,[_|T]):- member(H,T).
    unik([],[]). unik([H|T],L):-member(H,T),!,unik(T,L). unik([H|T],[H|L]):-
unik(T,L).
    listopt(Beg1,End1,Beg2,End2,Beg3,End3):-str_goal,shiftwindow(8),
clearwindow,
    write(«Доступные цели:\nЦели первого уровня:\n»),s_list0(L1),
unik(L1,L),
        cursor(Beg1,_), writelist(L),cursor(E1,_),End1=E1-1,
```

```
        write(«Цели второго уровня:\n»),s_list1(L2),unik(L2,L3),
        cursor(Beg2,_),writelist(L3),cursor(E2,_),End2=E2-1,
        write(«Цели третьего уровня:\n»),s_list2(L4),unik(L4,L5),
        cursor(Beg3,_),writelist(L5),cursor(E3,_),End3=E3-1.
    evalans("д"):-write(« да»),write(«\nЭкспертная система завершила работу.\n»),
     shiftwindow(3),write(« да»),write(«\nЭкспертная система завершила работу.\n»),
    shiftwindow(5),clearwindow,lst(L),how(L),readchar(_), shiftwindow(3),
    shiftwindow(2),clearwindow,!,clear,retract(lst(_)),assert(lst([])), menu(2).
    evalans(_):-write(« нет»),write(«\nЭкспертная система завершила работу\n»),
    shiftwindow(3),write(«нет»),write(«\nЭкспертная система завершила работу\n»),
    shiftwindow(2),clearwindow,!,clear,retract(lst(_)),assert(lst([])),menu(2).
    goes(_,Mygoal):-not(rule(_,Mygoal,_,_,_)),!,shiftwindow(2),
clearwindow,nl,
    write(«Наверное, это: «,Mygoal),nl,nl,write(«Правильно? (д/н)»),shiftwindow(3),
    write(«Наверное, это: «,Mygoal),write(«\nПравильно? (д/н)»),shiftwindow(2),
    cursorform(6,8),readchar(Ans),cursorform(14,15),evalans(Ans).
    /* Механизм вывода просматривает утверждения базы знаний rule и cond  для определения подходящих значений данных. Далее предикат check сопоставляет объекты, классифицированные номерами условий. Если сопоставление происходит, то в программу добавляются сопоставленные значения, и процесс продолжается с новыми данными, полученными от пользователя. Если сопоставления не происходит, то выбирается новая ветвь дерева поиска.*/
    goes(HISTORY,Mygoal):-rule(RNO,Mygoal,NY,COND,_),
      check(RNO,HISTORY, COND),%доказательство цели
      lst(L1),add(RNO,L1,L),retract(lst(_)),assert(lst(L)),!,
      goes([RNO|HISTORY],NY).%смена цели
    check(RNO,HISTORY,[BNO|REST]):-yes(BNO),!,check(RNO,HISTORY,REST).
    check(_,_,[BNO|_]):-no(BNO),!,fail.
    check(RNO,HISTORY,[BNO|REST]):-cond(BNO,NCOND),
```

```
      fronttoken(NCOND,»не»,_COND),frontchar(_COND,_,COND),
    cond(BNO1,COND),notest(BNO1),!,check(RNO,HISTORY,REST).
    check(_,_,[BNO|_]):-cond(BNO,NCOND),fronttoken(NCOND,
»не»,_COND),
      frontchar(_COND,_,COND),cond(BNO1,COND),yes(BNO1),!,fail.
    check(RNO,HISTORY,[BNO|REST]):-cond(BNO,TEXT),
      inpq(HISTORY,RNO,BNO,TEXT),check(RNO, HISTORY, REST).
    check(_,_,[]).
    inpq(HISTORY,RNO,BNO,TEXT):-shiftwindow(3),write(«Правда, что
«,TEXT,»? «),
      shiftwindow(2),clearwindow,cursor(4,0),write(«  Правда, что
«,TEXT,»? «),

menu2(CHOICE),do_answer(HISTORY,RNO,TEXT,BNO,CHOICE).
    notest(BNO):-no(BNO),!. notest(BNO):-not(yes(BNO)),!.
    key_menu2(27,X,X):-clearwindow,shiftwindow(3),nl,clear,retract(lst(_)),
      assert(lst([])),menu(2).
    key_menu2(75,X,Z):-field_attr(7,X,5,112),X>7,!,Z=X-6,cursor(7,Z),fail.
    key_menu2(77,X,Z):-field_attr(7,X,5,112),X<15,!,Z=X+6,cursor(7,Z),fail.
    key_menu2(13,X,X).
    menu2(CHOICE):- cursor(7,0),cursorform(14,15),write(«     Да   Нет
А смысл?»),
      cursor(7,7),rpt,cursor(_,X),      field_attr(7,X,5,113),inkey(Key),
      key_menu2(Key,X,CHOICE),!.
    do_answer(HISTORY,RNO,_,BNO,7):-assert(yes(BNO)),
shiftwindow(3),write(«да»),nl.
    do_answer(HISTORY,RNO,_,BNO,13):-assert(no(BNO)),
      shiftwindow(3),write(«нет»),nl,fail.
    do_answer(HISTORY,RNO,TEXT,BNO,19):- !,shiftwindow(5),
clearwindow,
      rule( RNO, Mygoal1, Mygoal2, _ ,_),sub_cat(Mygoal1,Mygoal2,Lstr),
      concat(«Попытка показать: «,Lstr,Lstr1),
      concat(Lstr1,»\nиспользуя правило «,Ls1),str_int(Str_num,RNO),
      concat(Ls1,Str_num,Ans),
show_rule(RNO,Lls1),concat(Ans,Lls1,Ans1),
      report(HISTORY,Sng),concat(Ans1,Sng,Answ),display(Answ),
    shiftwindow(3),shiftwindow(2),menu2(CHOICE),
```

```
        do_answer(HISTORY,RNO,TEXT,BNO,CHOICE).
    sub_cat(Mygoal1,Mygoal2,Lstr):-concat(Mygoal1,» это «,Str),concat(Str,
Mygoal2,Lstr).
    show_rule(RNO,Strg):-rule( RNO, Mygoal1, Mygoal2,
CONDINGELSER,_),
        str_int(RNO_str,RNO),concat(«\n Правило «,RNO_str,Ans),
        concat(Ans,»: «,Ans1),sub_cat(Mygoal1,Mygoal2,Lstr),
        concat(Ans1,Lstr,Ans2),concat(Ans2,»\n    если «,Ans3),

reverse(CONDINGELSER,CONILS),show_conditions(CONILS,Con),
        concat(Ans3,Con,Strg).
    report([],»»).
    report([RNO|REST],Strg) :-rule( RNO, Mygoal1, Mygoal2, _,_),
        sub_cat(Mygoal1,Mygoal2,Lstr),concat(«\nПоказано, что: «,Lstr,L1),
        concat(L1,»\nиспользуя правило «,L2),str_int(Str_RNO,RNO),
        concat(L2,Str_RNO,L3),concat(L3,»:\n «,L4),
show_rule(RNO,Str),

concat(L4,Str,L5),report(REST,Next_strg),concat(L5,Next_strg,Strg).
    show_conditions([],»»).
    show_conditions([COND],Ans):-show_cond(COND,Ans),!.
    show_conditions([COND|REST],Ans):-show_cond(COND,Text),
            concat(«\n    и «,Text,Nstr),show_conditions(REST,Next_ans),
             concat(Next_ans,Nstr,Ans).
    show_cond(COND,TEXT):-cond(COND,TEXT).
    reverse(X,Y):-reverse1([],X,Y).  reverse1(Y,[],Y).
    reverse1(X1,[U|X2],Y):-reverse1([U|X1],X2,Y).
    how(HISTORY):- report(HISTORY,Sng),concat(«»,Sng,Answ),
display(Answ).
    list                                                    :-
shiftwindow(6),cursorform(6,8),findall(RNO,rule(RNO,_,_,_,_),LIST),
    llist(List,Str),!,display(Str),!,cursorform(14,15),readchar(_),clearwindow,
shiftwindow(3),
            menu(6).
    llist([],»») :-!.  llist([RNO|List],Str):-llist(List,Oldstr),show_rule
(RNO,RNO_Str),
            concat(RNO_Str,Oldstr,Str).
```

```
save_y("д",D,Filename):-openwrite(save_file,Filename),
writedevice(save_file),
        write(D),closefile(save_file). save_y(_,_,_).
edit_rule:-shiftwindow(7),cursorform(6,8),file_str(«Mobile1.dbf»,Data),
edit(Data,NewData),clearwindow,write(«Сохранить набор правил (д/н) «),
readchar(Ans),save_y(Ans,NewData,»Mobile1.dbf»),cursorform(14,15),
  shiftwindow(3),menu(8).
GOAL
 cursorform(14,15),
 makewindow(8,112,112,»Определение цели»,15,0,34,80),
 makewindow(7,112,112,»Редактирование правил»,15,0,34,80),
 makewindow(6,112,112,»Просмотр правил»,15,0,34,80),
 makewindow(5,112,112,»Как была доказана цель»,15,0,34,80),
 makewindow(4,112,112,»Пополнение»,0,40,15,40),
 makewindow(3,112,112,»Результаты»,15,0,34,80),
 makewindow(2,112,112,»Экспертная часть»,0,40,15,40),
 makewindow(1,112,112,»:Животные»,0,0,15,40),
 assert(s_list0([])),assert(s_list1([])),assert(s_list2([])),
  menu(2).
```

Рассмотрим работу механизма вывода для идентификации животного с характеристиками в базе знаний [2,12,13,15] (Гепард).

В пользовательском интерфейсе в меню выбирается пункт "Запуск программы" (предикат run), в котором пользователь вводит, например, категорию Хищные и который вызывает предикат goes – начальное правило механизма вывода.

Переменная Mygoal получает значение Хищные. Применяется первое утверждение базы знаний на втором уровне:

```
rule(17,»Хищные»,»Тигр»,[13,14],2).
```

Переменная COND (список) получает значение [13]. Затем правило rule передает этот параметр правилу check, которое осуществляет доступ к правилу cond базы знаний с параметром BNO=13. Далее правило check осуществляет доступ к значению «Имеют рыжевато-коричневый цвет» и передает его в переменной TEXT правилу inpq. Правило inpq выдает на экран строку:

Вопрос: Имеют рыжевато-коричневый цвет ?.

Пользователь в меню должен выбрать Да или Нет. Правило inpq принимает ответ пользователя (Да) и процесс продолжается со следую-

щим значением COND=14.

Вопрос: Имеют черные полосы? Нет

Процесс продолжается со следующим правилом базы знаний (RNO =18).

Вопрос: Имеют темные пятна ? Да

Необходимый набор характеристик животного COND=[13,15]

при сопоставлении со списками условий в правилах связывается с объектом Гепард.

Ответ на вопрос Почему

Вопрос ПОЧЕМУ возникает в ситуации, когда пользователь желает знать, почему те или иные факты необходимы системе.

Допустим, что система спрашивает: Кто-то-там ест мясо?

Пользователь может спросить: Почему?

Система должна объяснить, почему ей нужен этот факт:

Потому, что, используя этот факт, она может вывести по правилу 14, что Кто-то-там – либо хищное, либо нет.

Система демонстрирует то, как она намерена использовать информацию от пользователя

Ответ на вопрос Как

Формирование ответа на вопрос Как представляет дерево подцелей, которые использовались для достижения полученного заключения. Например:

С: Кто-то-там - это Тигр, хотите узнать, Как?

П: Да.

С: Кто-то-там - это Тигр
выведено по правилу 17 из
Кто-то-там - это хищное
выведено по правилу 14 из
Кто-то-там - это млекопитающие
выведено по правилу 2 из фактов:
Кто-то-там имеет шерсть,
Кто-то-там ест мясо,
Кто-то-там имеет рыжевато-коричневый цвет,
Кто-то-там имеет черные полосы.

Представление экспертной системы

Визуально функционирование экспертной системы отображается на приведенных рисунках.

На рисунке 5 показан редактор, включающий меню Животные, экспертную часть и определение цели. При запуске ЭС на поле Определение цели отображаются все уровни понятий базы знаний, которые можно выбрать пользователю.

Рис. 5

На рис.6 показан процесс идентификации животного Гепарда.

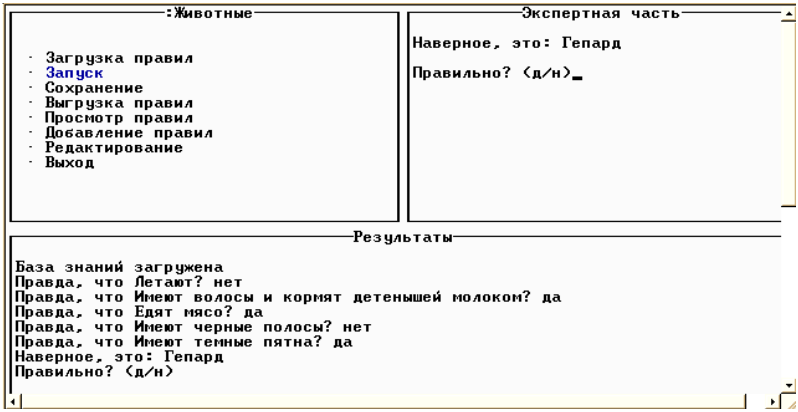


Рис. 6

На рисунке 7 показан ответ ЭС на вопрос: Почему.

Базу знаний можно дополнить правилами, уточняющими идентификацию животных и добавляющими новые понятия (категории) и объекты на любом уровне (рисунок 8).

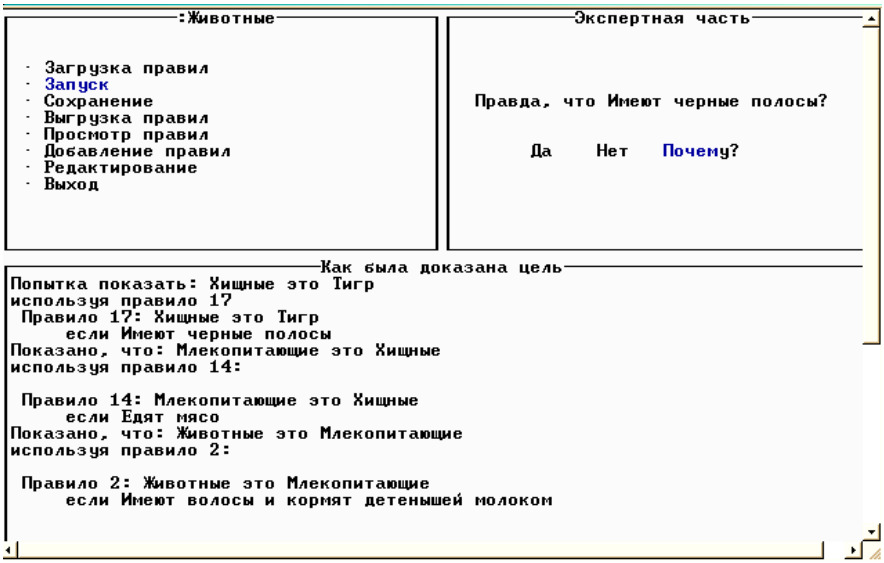


Рис. 7

Рис. 8

При работе с экспертной системой возможен просмотр правила базы знаний и их редакция (рисунок 9).

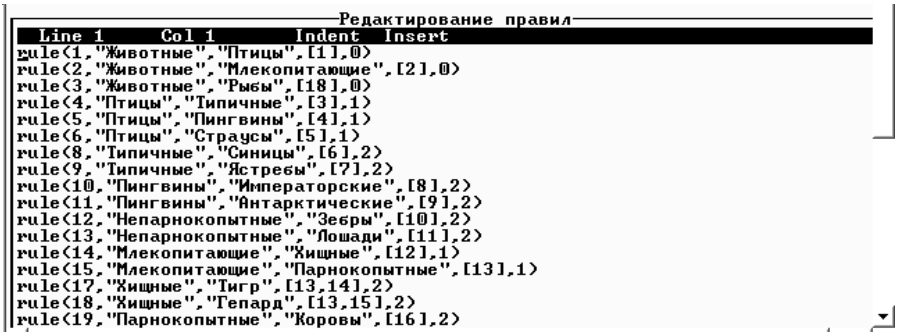


Рис. 9

НЕКОТОРЫЕ ПРИЛОЖЕНИЯ ТУРБО ПРОЛОГА

Задача обработки маршрута

С помощью Турбо Пролога возможно нахождение различных маршрутов между городами.

Фактами являются расстояния между городами.

Рассмотрим упрощенную схему автомобильных дорог межу некоторыми городами (рисунок 10).

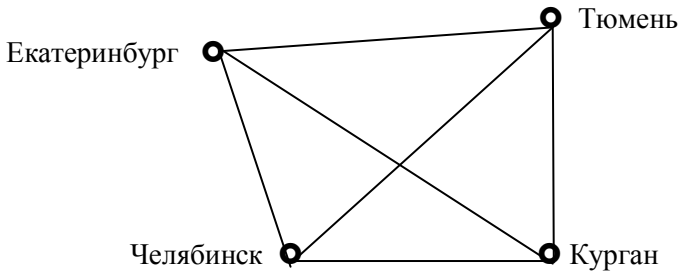


Рис. 10

Программа:

```
domains
   town = symbol
   distance = integer
```

```
predicates
   road(town, town, distance)
   route(town, town, distance)
clauses
        road("Курган", "Екатеринбург", 360).
        road("Челябинск", "Курган", 240).
        road("Челябинск", "Екатеринбург", 250).
        road("Челябинск", "Тюмень", 400).
        road("Курган", "Тюмень", 210).
        road("Екатеринбург", "Тюмень", 300).
        route(Town1, Town2, Distance):- road(Town1,Town2,
                                                Distance).
          route(Town1, Town2, Distance):- road(Town1, X, Dist1),
              route(X, Town2, Dist2),
               Distance = Dist1 + Dist2.
```

Программа представляет собой базу данных. В каждом предложении для предиката road (дорога) описывается дорога, ведущая от одного города к другому, и указывается расстояние в километрах.

Предложения для предиката route (маршрут) говорят о том, что маршрут от одного города до другого может состоять из нескольких участков дороги через промежуточные города. Следуя по маршруту можно определять общее расстояние (distance). Для этого предикат route определяется рекурсивно.

Для цели: route("Челябинск", "Тюмень", X) Турбо Пролог выдаст решения: X = 400
X = 450
X = 900
X = 550

Интерес представляет задача составления маршрута по городам таким образом, чтобы избежать посещения городов дважды, а также выбора оптимального маршрута в соответствии с категорией дорог.

Поиск пути в лабиринте

В качестве примера использования механизма логического вывода рассмотрим процедуру поиска пути в лабиринте. Лабиринт представлен фактами вида: стена(I, J) для позиции в I-м ряду и J-й колонке, где есть стена; нет_стена(I, J) для позиции в I-м ряду и J-й колонке, где нет сте-

ны; выход (I, J) для позиции в I-м ряду и J-й колонке, являющейся выходом, вход (I, J) для позиции в I-м ряду и J-й колонке, являющейся входом.

Рассмотрим небольшой лабиринт (рис.11).

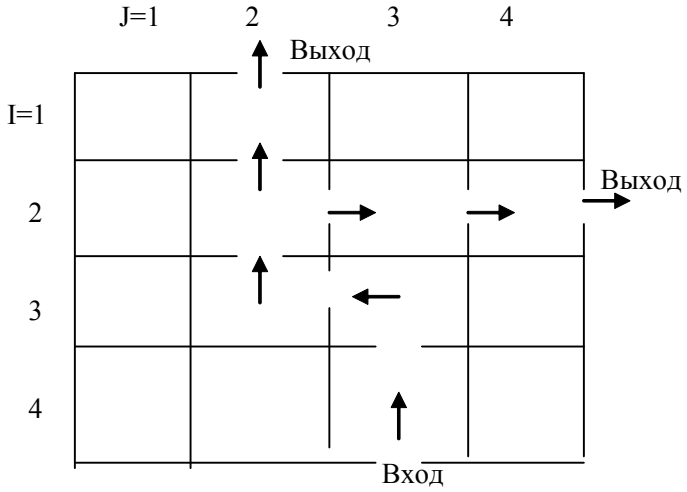


Рис. 11

Лабиринт описывается фактами:

стена(1,1). стена(1,3). стена(1,4).
стена(2,1).
стена(3,1). стена(3,4).
стена(4,1). стена(4,2). стена(4,4).
нет_стена(1,2).
нет_стена(2,2). нет_стена(2,3). нет_стена(2,4).
нет_стена(3,2). нет_стена(3,3).
нет_стена(4,3).
вход(4,3). выход(1,2). выход(2,4).

Граничное условие: если исходная позиция является выходом, то путь найден.

Алгоритм поиска выхода из лабиринта

Из исходной позиции (Входа) идти в прямом направлении. Если стены нет - идти прямо, если стена - идти налево. Если налево стена - идти

направо. Если направо стена - идти обратно. Чтобы не ходить по кругу, вести список позиций, которые посещались.

Для поиска выхода строится соответствующая процедура пути: она ищет из некоторой позиции (первый аргумент) путь (второй аргумент) к выходу. Третьим аргументом является список посещаемых позиций.

Вводится терм a(I, J), который представляет позицию в I-м ряду и J-й колонке.

Граничное условие:

путь(a(I, J),[a(I, J)], Были) :- выход(I, J).

Реализация алгоритма:

Идти прямо:

путь(a(I, J),[a(I, J) | P], Были):-K = I-1,
идти(a(K, J), Были), путь(a(I, J) ,P, [a(K, J)| Были]).

Идти налево:

путь(a (I, J), [a(I, J) | P], Были):-L = J-1,
идти(a(I, L), Были), путь(a(I, L), P, [a(I, L)| Были]).

Идти направо:

путь(a (I, J), [a(I, J) | P], Были):-L = J+1,
идти(a(I, L), Были), путь(a(I, L), P, [a(I, L)| Были]).

Идти обратно:

путь(a(I, J),[a(I, J) | P], Были):-K = I+1,
идти(a(K, J), Были), путь(a(I, J) ,P, [a(K, J)| Были]).

В позицию a(I, J) можно попасть при условии, что там нет стены и она не посещалась прежде.

идти(a(I, J)), Были):-нет_стена(I, J),
not (принадлежит (a (I, J), Были)).

Программа:

```
domains
   a=a(integer,integer)
    pos=a*
predicates
   stena(integer,integer)
   nstena(integer,integer)
   put(a,pos,pos)
   idti(a,pos)
    member(a,pos)
clauses
```

```
put(a(I,J),[a(I,J)],B).
put(a(I,J),[a(I,J)|P],B):-K=I-1,
                          idti(a(K,J),B),put(a(K,J),P,[a(K,J)|B]).
put(a(I,J),[a(I,J)|P],B):-K=I+1,
                          idti(a(K,J),B),put(a(K,J),P,[a(K,J)|B]).
put(a(I,J),[a(I,J)|P],B):-L=J-1,
                           idti(a(I,L),B),put(a(I,L),P,[a(I,L)|B]).
put(a(I,J),[a(I,J)|P],B):-L=J+1,
                           idti(a(I,L),B),put(a(I,L),P,[a(I,L)|B]).
idti(a(I,J),B):-nst(I,J),not(member(a(I,J),B)).
member(X,[X|_]):-!.
member(X,[_|T]):-member(X,T).
stena(1,1).stena(1,3).stena(1,4).stena(2,1).stena(3,1).
stena(3,4).stena(4,1).stena(4,2).stena(4,4).
nstena(1,2).nstena(2,2).nstena(2,3).nstena(2,4).
nstena(3,2).nstena(3,3).nstena(4,3).
```

Зададим путь от входа в лабиринт - a(4,3).

Цель: put(a(4,3),P,[a(4.3)]).

Получим варианты выхода из лабиринта:

P = [a(4,3), a(3,3), a(3,2), a(2,2), a(1,2)]

P = [a(4,3), a(3,3), a(3,2), a(2,2), a(2,3), a(2,4)]

Интересным является усложнение задачи, например, для поиска из множества вариантов выхода из лабиринта оптимального варианта.

Моделирование аппаратуры

Всякую логическую схему можно описать с помощью предикатов Турбо Пролога, задав отношения между внутренними связями, а также между входами и выходами схемы.

Рассмотрим схему для исключающего ИЛИ, приведенную на рисунке 12, построенную с помощью логических связок И, ИЛИ, НЕ.

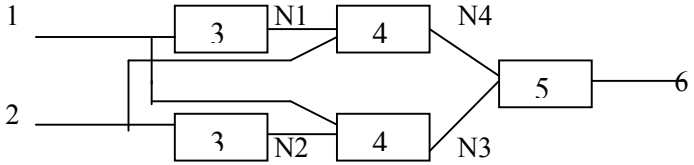


Рис. 12

1 – вход1, 2 – вход2, 3 – НЕ, 4 – И, 5 – ИЛИ, 6 – выход.

В приведенной ниже программе эта цепь описывается с помощью предиката xor (исключающее ИЛИ).

```
domains
   d = integer
predicates
   not_(D, D)
   and_(D, D, D)
   or_(D, D, D)
   xor(D, D, D)
clauses
   not_(1, 0). not_(0, 1).
   and_(0, 0, 0). and_(0, 1, 0). and_(1, 0, 0). and_(1, 1, 1).
   or_(0, 0, 0). or_(0, 1, 1). or_(1, 0, 1). or_(1, 1, 1).
   xor(Input1, Input2, Output):-
                              not_(Input1, N1), not_(Input2, N2),
                              and_(Input1, N2, N3), not_(Input2, N4),
                               or_(N3, N4, Output).
```

Цель: xor(Input1, Input2, Output)

Турбо Пролог выведет следующий результат:

```
Input1 = 1,  Input2 = 1,  Output = 0
Input1 = 1,  Input2 = 0,  Output = 1
Input1 = 0,  Input2 = 1,  Output = 1
Input1 = 0,  Input2 = 0,  Output =0
```

Программа раскрывает логику работы исследуемой схемы. Задав множество взаимосвязанных логических элементов, можно получить на этом множестве разнообразные логические схемы в соответствии с необходимыми требованиями.

Команды редактора

Таблица 1

Значение	Управляющие клавиши	Дополнительные клавиши
Команды перемещения курсора		
На символ влево	Ctrl-S	←
На символ вправо	Ctrl-D	→
На слово влево	Ctrl-A	Ctrl-←
На слово вправо	Ctrl-F	Ctrl-→
Вверх на строку	Ctrl-E	↑
Вниз на строку	Ctrl-X	↓
Вверх на страницу	Ctrl-R	PgUp
Вниз на страницу	Ctrl-C	PgDn
К началу строки	Ctrl-QS	Home
К концу строки	Ctrl-QD	End
К началу файла	Ctrl-QR	Ctrl-PgUp
К концу файла	Ctrl-QC	Ctrl-PgDn
Команды стирания		
Режим вставки/замены	Ctrl-V	Ins
Стереть левый символ		Backspace
Стереть символ курсора	Ctrl-G	Del
Стереть строку	Ctrl-Y	Ctrl-Backspace
Стереть символы от курсора до конца строки	Ctrl-QY	
Команды работы с блоками		
Отметить начало блока	Ctrl-KB	
Отметить конец блока	Ctrl-KK	
Скопировать блок	Ctrl-KC	F5
Передвинуть блок	Ctrl-KV	F6
Стереть блок	Ctrl-KY	F7
Считать блок с диска	Ctrl-KR	F9
Записать блок на диск	Ctrl-KW	
Включить/выключить маркировку блока	Ctrl-KH	
Прочие команды		
Вызвать справку по редактору		F8
Поиск	Ctrl-QF	F3
Поиск и замена	Ctrl-QA	F4

Список стандартных предикатов

asserta(Факт) – Добавление факта в начало БД
assertz(Факт) - Добавление факта в конец БД
attribute(Атрибут) – Установка атрибута
back(Шаг) – Продвижение пера в обратном направлении
beep – Звуковой сигнал
bound(Переменная) – Проверка, связана ли переменная
char_int(Символ,Число) – Связывание символа с числом кода ASCII
clearwindow - Очистка текущего окна
closefile(ИмяФайла) – Закрытие файла
comline(Строка) – Чтение параметров командной строки
concat(Строка1,Строка2,Строка3) – Склеивание двух строк
consult(Имя ФайлаDOS) – Добавление файла в БД
cursor(Строка, Столбец) – Позиционирование курсора
cursorform(НачСтрока,КонСтрока) – Определение курсора
cut(!) – Предотвращение бектрекинга
date(Год,Месяц,День) – Установка и считывание даты
deletefile(ИмяФайлаDOS) – Удаление файла
dir(Путь,Спецификация файла,ИмяФайлаDOS) – Вывод текущего каталога
disk(ПутьDOS) – Установка пути и накопителя
display(Строка) – Показывание строки
dot(Строка,Столбец,Цвет) – Установка цвета точки
edit(ВхСтрока,ВыхСтрока) – Вызов редактора Турбо Пролога
eof(ИмяФайла) – Проверка на конец файла
existfile(ИмяФайлаDOS) – Проверка, существует ли файл
exit – Окончание программы
fail – Инициирование бектрекинга
field_attr(Строка,Столбец,Длина,ЗначАтр) – Установка атрибута поля
field_str(Строка,Столбец,Длина,СтрСимволов) – Запись или чтение строки
filemode(ИмяФайла,ТипФайла) – Установка типа файла
filepos(ИмяФайла,Позиция,Режим) – Установка позиции указателя
file_str(ИмяФайлаDOS,Строка) – Чтение строки из файла
findall(Переменная,Атом,Список)– Сбор значений в список после бектрекинга
flush(ИмяФайла) – Очистка содержимого буфера
forward(Шаг) – Продвижение пера вперед
free(Переменная) – Проверка, свободна ли переменная

frontchar(Строка,ПервСимвол,Остаток) – Разделяет строку на две части
frontstr(КолСим,ВхСтр,ВыхСтр,Остаток) – Разделяет строку в задан. позиции
fronttoken(Стр,Лексема,Остаток) – Разделяет строку на лексему и остаток
gotowindow(НомОкна) – Выбор окна
graphics(Режим,Палитра,Фон) – Установка графического режима
inkey(Символ) – Чтение символа
keypressed – Проверка, нажата ли клавиша
left(Угол) – Поворот пера влево
line(Строка1,Столбец1,Строка2,Столбец2,Цвет) – Рисование линии
makewindow(НомОкна,АтрЭкр,АтрРамки,Заголовок,Строка,Столбец,Высота, Ширина) – Определение окна
nl – Перевод строки
not(Атом) - Отрицание
openappend(ИмяФайла,ИмяФайлаDOS) – Открытие файла для дополнения
openmodify(ИмяФайла,ИмяФайлаDOS) – Открытие файла для чтения/записи
openread(ИмяФайла,ИмяФайлаDOS) – Открытие файла для чтения
openwrite(ИмяФайла,ИмяФайлаDOS) – Открытие файла для записи
pencolor(Цвет) – Установка цвета линии, проводимой пером
pendown – Активация пера
penpos(Строка,Столбец,Направление) – Установка позиции пера
penup – Деактивация пера
portbyte(НомерПорта,Значение) – Посылка байта в порт или чтение из порта
readchar(СимвПеременная) – Чтение символа с устройства ввода
readdevice(ИмяФайла) – Определение символич. имени устройства ввода
readint(ЦелаяПеременная) – Чтение целого числа
readln(Строка) – Чтение строки
readreal(ВещПеременная) – Чтение действительного числа
readterm(Область,Терм) – Чтение объектов, записанных предикатом write
removewindow – Удаление текущего окна
renamefile(СтарИмяФайлаDOS,НовИмяФайлаDOS) – Переименование файла
retract(Факт) – Удаление факта из БД

right(Угол) – Поворот пера вправо
save(ИмяФайлаDOS) – Запись на диске БД
scr_attr(Строка,Столбец,Атр) – Установка атрибута
scroll(ЧислоСтрок,ЧислоСтолбцов) – Сдвиг содержимого текущего окна
shiftwindow(НомОкна) – Смена текущего окна
sound(Продолжительность,Частота) – Производство звукового сигнала
storage(Стек,ДинПам,Остаток) – Определение размера имеющейся памяти
system(Команда DOS) – Обеспечение выполнения команд DOS
text – Установка текстового режима
window_attr(Атрибут) – Определение атрибутов текущего окна
window_str(Строка) – Запись строки в текущее окно
write(A1,A2,…) – Вывод на экран переменных и констант
writedevice(СимвИмяФайла) – Определение символич. имени файла вывода
writef(Формат,A1,A2,…) – Осуществление форматного вывода

СПИСОК ЛИТЕРАТУРЫ

1. Доорс Дж. и другие. Пролог – язык программирования будущего. - М., 1990.
2. Малпас Дж. Реляционный язык Пролог и его применение. - М., 1990.
3. Марселлус Д. Программирование экспертных систем на Турбо-Прологе.- М.: 1994.
4. Чень Ч., Ли Р. Математическая логика и автоматическое доказательство теорем. - М.: Наука, - 1983
5. Макаллистер Дж. Искусственный интеллект и Пролог на микроЭВМ. - М., 1990.
6. Братко И. Программирование на языке Пролог для искусственного интеллекта. - М., 1990.
7. Ин Ц., Соломон Д. Использование Турбо Пролога. – М.: Мир. – 1990.
8. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. - М., 1990.
9. Чери С. и другие. Логическое программирование и базы данных. – М., 1992.
10. Грэй П. Логика, алгебра и базы данных. – М., 1989.
11. Гаврилова Т.А., Червинская К.Р. Извлечение и структурирование знаний для экспертных систем. – М.: Радио и связь. –1992.
12. Таунсенд К., Фохт Д. Проектирование и программная реализация экспертных систем на персональных ЭВМ. – М.: Финансы и статистика, - 1990.

Учебное издание

Кокин Александр Георгиевич

ТУРБО ПРОЛОГ

Учебное пособие

Редактор Н.М. Кокина

Подписано в печать	Формат 60x84 1/16	Бумага тип. №1
Печать трафаретная	Усл.печ.л. 6,0	Уч.-изд.л. 6,0
Заказ	Тираж 150	Цена свободная

Редакционно-издательский центр КГУ.
640669, г. Курган, ул. Гоголя, 25.
Курганский государственный университет.