МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

Методические указания
по выполнению лабораторных работ по дисциплине
«ВВЕДЕНИЕ В ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ»

направления подготовки **09.03.02 Информационные системы и технологии**

Направленность (профиль): **Информационные системы и технологии обработки цифрового контента**

Пятигорск, 2025г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В методических указаниях содержатся материалы, необходимые для самостоятельной подготовки студентов к выполнению лабораторных работ. В описание лабораторных работ включены цель работы, порядок ее выполнения, рассмотрены теоретические вопросы, связанные с реализацией поставленных задач, приведена необходимая литература.

Методические указания посвящены курсу «Введение в функциональное программирование». На протяжении многих тысячелетий человечество занимается накоплением, обработкой и передачей знаний. Для этих целей непрерывно изобретаются новые средства и совершенствуются старые: речь, письменность, почта, телеграф, телефон и т. д. Большую роль в технологии обработки знаний сыграло появление компьютеров.

В октябре 1981 года Японское министерство международной торговли и промышленности объявило о создании исследовательской организации — Института по разработке методов создания компьютеров нового поколения (Institute for New Generation Computer Technology Research Center). Целью данного проекта было создание систем обработки информации, базирующихся на знаниях. Предполагалось, что эти системы будут обеспечивать простоту управления за счет возможности общения с пользователями при помощи естественного языка. Эти системы должны были самообучаться, использовать накапливаемые в памяти знания для решения различного рода задач, предоставлять пользователям экспертные консультации, причем от пользователя не требовалось быть специалистом в информатике. Предполагалось, что человек сможет использовать ЭВМ пятого поколения так же легко, как любые бытовые электроприборы типа телевизора, магнитофона и пылесоса. Вскоре вслед за японским стартовали американский и европейский проекты.

Появление таких систем могло бы изменить технологии за счет использования баз знаний и экспертных систем. Основная суть качественного перехода к пятому поколению ЭВМ заключалась в переходе от обработки данных к обработке знаний. Японцы надеялись, что им удастся не подстраивать мышление человека под принципы функционирования компьютеров, а приблизить работу компьютера к тому, как мыслит человек, отойдя при этом от фон неймановской архитектуры компьютеров. В 1991 году предполагалось создать первый прототип компьютеров пятого поколения.

Теперь уже понятно, что поставленные цели в полной мере так и не были достигнуты, однако этот проект послужил импульсом к развитию нового витка исследований в области искусственного интеллекта и вызвал взрыв интереса к логическому программированию. Так как для эффективной реализации традиционная фон неймановская архитектура не подходила, были созданы специализированные компьютеры логического программирования PSI и PIM.

В качестве основной методологии разработки программных средств для проекта ЭВМ пятого поколения было избрано логическое программирование, ярким представителем которого является язык Пролог. Думается, что и в настоящее время Пролог остается наиболее популярным языком искусственного интеллекта в Японии и Европе (в США, традиционно, более распространен другой язык искусственного интеллекта —язык функционального программирования Лисп).

Название языка "Пролог" происходит от слов ЛОГическое ПРОграммирование (PROgrammation en LOGique во французском варианте и PROgramming in LOGic — в английском).

Пролог основывается на таком разделе математической логики, как исчисление предикатов. Точнее, его базис составляет процедура доказательства теорем методом резолюции для хорновских дизъюнктов.

1. ЦЕЛЬ И ЗАДАЧИ ИЗУЧЕНИЯ ДИСЦИПЛИНЫ

Цель освоения дисциплины «Введение в функциональное программирование» формирование набора профессиональных компетенций будущего бакалавра по направлению подготовки 09.03.02 «Информационные системы и технологии».

Задачи освоения дисциплины: изучение основных понятий функционального программирования, освоение инструментов решения задач функционального программирования.

2. ОБОРУДОВАНИЕ И МАТЕРИАЛЫ

Аппаратные средства: персональный компьютер;

Программные средства Альт Рабочая станция 10, Альт Рабочая станция K, Альт «Сервер», Пакет офисных программ - Р7-Офис.

Учебный класс оснащен IBM-совместимыми компьютерами, объединенными в локальную сеть. Локальная сеть учебного класса имеет постоянный доступ к сети Internet по выделенной линии. Для проведения лабораторных работ необходимо следующее программное обеспечение: операционная система Альт Рабочая станция, пакет офисных программ Р7-Офис.

3. УКАЗАНИЯ ПО ТЕХНИКЕ БЕЗОПАСНОСТИ

Перед началом работы следует убедиться в исправности электропроводки, выключателей, штепсельных розеток, при помощи которых оборудование включается в сеть, наличии заземления компьютера, его работоспособности.

Для снижения или предотвращения влияния опасных и вредных факторов необходимо соблюдать санитарные правила и нормы, гигиенические требования к персональным электронно-вычислительным машинам.

Во избежание повреждения изоляции проводов и возникновения коротких замыканий не разрешается: вешать что-либо на провода, закрашивать и белить шнуры и провода, закладывать провода и шнуры за газовые и водопроводные трубы, за батареи отопительной системы, выдергивать штепсельную вилку из розетки за шнур, усилие должно быть приложено к корпусу вилки.

Для исключения поражения электрическим током запрещается: часто включать и выключать компьютер без необходимости, прикасаться к экрану и к тыльной стороне блоков компьютера, работать на средствах вычислительной техники и периферийном оборудовании мокрыми руками, работать на средствах вычислительной техники и периферийном оборудовании, имеющих нарушения целостности корпуса, нарушения изоляции проводов, неисправную индикацию включения питания, с признаками электрического напряжения на корпусе, класть на средства вычислительной техники и периферийном оборудовании посторонние предметы.

Запрещается под напряжением очищать от пыли и загрязнения электрооборудование.

Во избежание поражения электрическим током, при пользовании электроприборами нельзя касаться одновременно каких-либо трубопроводов, батарей отопления, металлических конструкций, соединенных с землей.

После окончания работы необходимо обесточить все средства вычислительной техники и периферийное оборудование. В случае непрерывного учебного процесса необходимо оставить включенными только необходимое оборудование.

4. СОДЕРЖАНИЕ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа 1. Введение в программирование на языке PROLOG

Цель работы: изучение инструментов логического программирования, изучение SWI-PROLOG. Создание и запуск программ на PROLOG. Работа с интерпретатором SWI-PROLOG.

Основы теории

Императивные и декларативные языки программирования

Традиционно под программой понимают последовательность операторов (команд, выполняемых компьютером). Этот стиль программирования принято называть императивным. Программируя в императивном стиле, программист должен объяснить компьютеру, как нужно решать задачу.

Противоположный ему стиль программирования — так называемый декларативный стиль, в котором программа представляет собой совокупность утверждений, описывающих фрагмент предметной области или сложившуюся ситуацию. Программируя в декларативном стиле, программист должен описать, что нужно решать.

Соответственно и языки программирования делят на императивные и декларативные.

Императивные языки основаны на фон неймановской модели вычислений компьютера. Решая задачу, императивный программист вначале создает модель в некоторой формальной системе, а затем переписывает решение на императивный язык программирования в терминах компьютера. Но, во-первых, для человека рассуждать в терминах компьютера довольно неестественно. Во-вторых, последний этап этой деятельности (переписывание решения на язык программирования) по сути дела не имеет отношения к решению исходной задачи. Очень часто императивные программисты даже разделяют работу в соответствии с двумя описанными выше этапами. Одни люди, постановщики задач, придумывают решение задачи, а другие, кодировщики, переводят это решение на язык программирования.

В основе декларативных языков лежит формализованная человеческая логика. Человек лишь описывает решаемую задачу, а поиском решения занимается императивная система программирования. В итоге получаем значительно большую скорость разработки приложений, значительно меньший размер исходного кода, легкость записи знаний на декларативных языках, более понятные, по сравнению с императивными языками, программы.

Известна классификация языков программирования по их близости либо к машинному языку, либо к естественному человеческому языку. Те, что ближе к компьютеру, относят к языкам низкого уровня, а те, что ближе к человеку, называют языками высокого уровня. В этом смысле декларативные языки можно назвать языками сверхвысокого или наивысшего уровня, поскольку они очень близки к человеческому языку и человеческому мышлению.

К императивным языкам относятся такие языки программирования, как Паскаль, Бейсик, Си и т. д. В отличие от них, Пролог является декларативным языком.

Программирование на языке Пролог

При программировании на Прологе усилия программиста должны быть направлены на описание логической модели фрагмента предметной области решаемой задачи в терминах объектов предметной области, их свойств и отношений между собой, а не деталей программной реализации. Фактически Пролог представляет собой не столько язык для программирования, сколько язык для описания данных и логики их обработки. Программа на Прологе не является таковой в классическом понимании, поскольку не содержит явных управляющих конструкций типа условных операторов, операторов цикла и т. д. Она представляет собой модель фрагмента предметной области, о котором идет

речь в задаче. И решение задачи записывается не в терминах компьютера, а в терминах предметной области решаемой задачи, в духе модного сейчас объектно-ориентированного программирования.

Пролог очень хорошо подходит для описания взаимоотношений между объектами. Поэтому Пролог называют реляционным языком. Причем "реляционность" Пролога значительно более мощная и развитая, чем "реляционность" языков, используемых для обработки баз данных. Часто Пролог используется для создания систем управления базами данных, где применяются очень сложные запросы, которые довольно легко записать на Прологе.

В Прологе очень компактно, по сравнению с императивными языками, описываются многие алгоритмы. По статистике, строка исходного текста программы на языке Пролог соответствует четырнадцати строкам исходного текста программы на императивном языке, решающем ту же задачу. Пролог-программу, как правило, очень легко писать, понимать и отлаживать. Это приводит к тому, что время разработки приложения на языке Пролог во многих случаях на порядок быстрее, чем на императивных языках. В Прологе легко описывать и обрабатывать сложные структуры данных. Проверим эти утверждения на собственном опыте при изучении данного курса.

Прологу присущ ряд механизмов, которыми не обладают традиционные языки программирования: сопоставление с образцом, вывод с поиском и возвратом. Еще одно существенное отличие заключается в том, что для хранения данных в Прологе используются списки, а не массивы. В языке отсутствуют операторы присваивания и безусловного перехода, указатели. Естественным и зачастую единственным методом программирования является рекурсия.

Для решения любой задачи есть оптимальный язык (языки) программирования. Многие задачи, хорошо решаемые императивными языками типа Паскаля и Си, плохо решаются на Прологе, и наоборот. Основные области применения Пролога:

быстрая разработка прототипов прикладных программ;

автоматический перевод с одного языка на другой;

создание естественно-языковых интерфейсов для существующих систем;

символьные вычисления для решения уравнений, дифференцирования и интегрирования;

проектирование динамических реляционных баз данных;

экспертные системы и оболочки экспертных систем;

автоматизированное управление производственными процессами;

автоматическое доказательство теорем;

полуавтоматическое составление расписаний;

системы автоматизированного проектирования, базирующееся на знаниях программное обеспечение;

организация сервера данных или, точнее, сервера знаний, к которому может обращаться клиентское приложение, написанное на каком-либо языке программирования.

Знакомство с интерпретатором SWI-PROLOG

В папке с установленным SWI-PROLOG войдите в директорию pl/bin, содержащую файл plwin.exe, и запустите его. На экране появится главное меню и главное (диалоговое) окно с приглашением SWI-PROLOG (см рис.1).

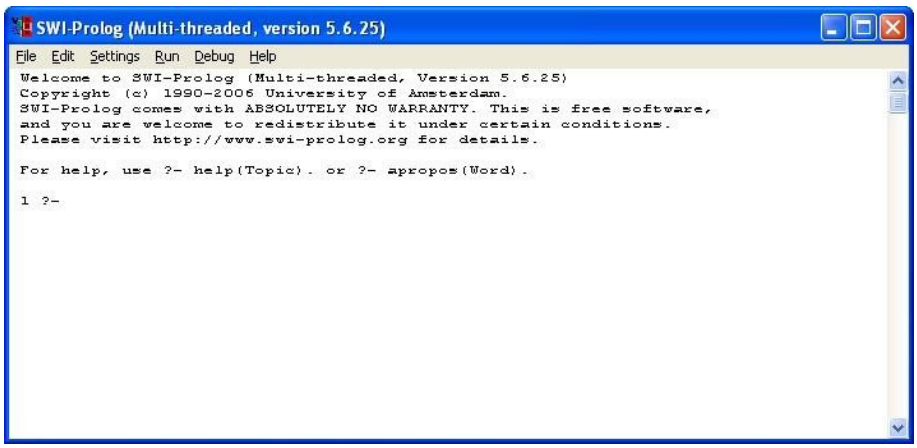

Рисунок 1.1 Вид диалогового окна SWI-PROLOG

Главное меню можно сделать активным, нажав F10 или Alt. Когда главное меню активно, его элементы можно выбрать с помощью клавиш управления курсором и последующим нажатием клавиши Enter. Выбирать элементы главного меню можно также и мышью.

Программа на Прологе

Программа на Прологе состоит из фактов и правил, которые образуют базу знаний Пролог-программы, и запроса к этой базе, который задает цель поиска решений.

Предикаты

Описывают отношение между объектами, которые являются аргументами предиката.

Факты

Констатируют наличие заданного предикатом отношения между указанными объектами.

ПРИМЕР

Констатация факта в предложении
Эллен любит теннис.

в синтаксисе Пролога выглядит так:

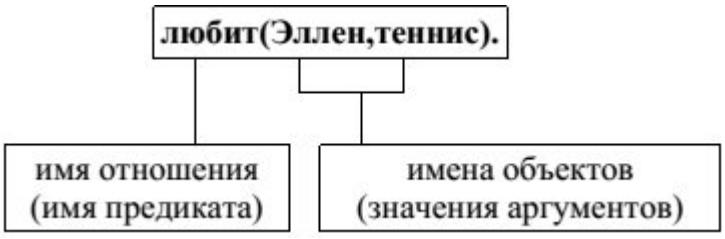

Рисунок 1.2 – Синтаксис Пролог

Имя предиката (функтора) и объекта должно начинаться с маленькой буквы и может содержать латинские буквы, кириллицу, цифры и символ подчеркивания (_). Кириллица используется наравне с латинскими буквами. Обычно предикатам дают такие имена, чтобы они отражали смысл отношения. Например: main, add_file_name. Два предиката могут иметь одинаковые имена, тогда система распознает их как разные предикаты, если они имеют различное число аргументов (арность). Например, любит/2, любит/3.

Имя предиката может совпадать с именем какого-либо встроенного предиката SWI-PROLOG. Однако, если совпали имена пользовательского и встроенного предиката, то при обращении к нему (либо из интерпретатора, либо из программы), будет вызван пользовательский предикат, т.е. пользовательское определение «перекроет» предопределенное в SWI-PROLOG.

Правила

Правила описывают связи между предикатами.

```
Билл любит все, что любит Том.
```

в синтаксисе Пролога:

```
любит('Билл', Нечто):- любит('Том',Нечто).
```

Правило B:-A соответствует импликации A→B («ЕСЛИ A , ТО B»).

В общем виде правило - это конструкция вида:

P0:-P1,P2,…,Pn.

которая читается «P0 истинно, если P1 и P2 и ... Pn истинны».

Предикат P0 называется заголовком правила, выражение P1,P2,…,Pn - телом правила, а предикаты Pi - подцелями правила. Запятая означает логическое "И".

Факты и правила называются также утверждениями или клозами. Факт можно рассматривать как правило, имеющее заголовок и пустое тело.

Процедура

Процедура это совокупность утверждений, заголовки которых имеют одинаковый функтор и одну и ту же арность. Процедура задает определение предиката.

Конец предложения всегда отмечается точкой, поэтому все факты, правила и запросы должны заканчиваться точкой. Заметим также, что между именем предиката и скобкой не должно быть пробелов.

Переменная

Переменная - поименованная область памяти, где может храниться значение.

Если переменная не связана со значением – она называется свободной переменной.

Унификация

Унификация - процесс получения свободной переменной значения в результате сопоставления при логическом выводе в SWI-PROLOG.

Понятие переменной в логическом программировании отличается от базового понятия переменной, которое вводится в структурном программировании. Прежде всего, это отличие заключается в том, что переменная в SWI-PROLOG, однажды получив свое значение при унификации в процессе работы программы, не может его изменить, т.е. она скорее является аналогом математического понятия «переменная» – неизвестная величина. Переменная в SWI-PROLOG не имеет предопределенного типа данных и может быть связана со значением любого типа данных.

Переменная в SWI/PROLOG обозначается как последовательность латинских букв, кириллицы и цифр, начинающаяся с заглавной буквы или символа подчеркивания (_). Заметим, что если значение аргумента предиката или его имя начинается с заглавной буквы, то оно пишется в апострофах.

В SWI-PROLOG различаются строчные и заглавные буквы.

Рассмотрим следующую программу на SWI-PROLOG, которую будем использовать для иллюстрации процессов создания, выполнения и редактирования Пролог-программ.

Листинг 1.1.

```
/* кто что любит */
```

```
любит('Эллен', теннис). %Эллен любит теннис
любит('Джон', футбол). %Джон любит футбол
любит('Том', бейсбол). %Том любит бейсбол
любит('Эрик', плавание). %Эрик любит плавание
любит('Марк', теннис). %Марк любит теннис
любит('Билл',X):-любит('Том',X).
%Билл любит то, что любит Том
```

Комментарий в строке программы начинается с символа % и заканчивается концом строки. Блок комментариев выделяется специальными скобками:
/* (начало) и */ (конец).

Задание 1.1

Для того чтобы набрать текст программы, воспользуйтесь встроенным текстовым редактором. Чтобы создать новый файл выберете команду File/New, в диалоговом окне укажите имя нового файла, например, тест. Для редактирования уже созданного файла с использованием встроенного редактора можно воспользоваться командой меню File/Edit. Набейте программу 1 (текст программы выше по тексту) и сохраните ее (File/Save buffer).

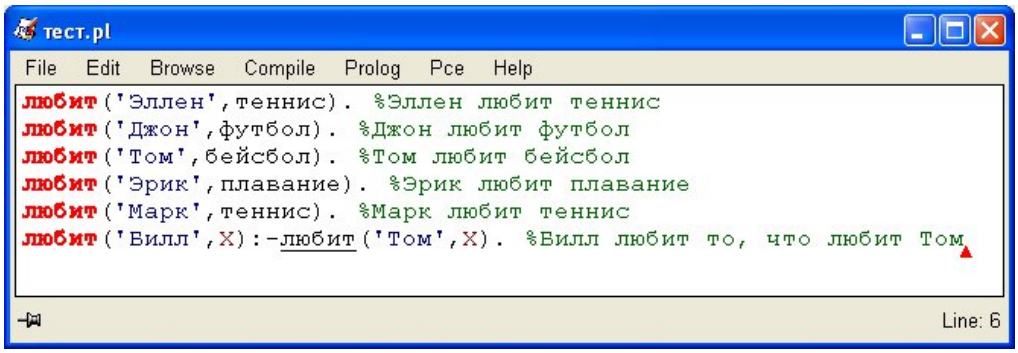

Рисунок 1.2 Внешний вид редактора

Красным цветом подсвечиваются предикаты в заголовках предложений, которые с точки зрения синтаксиса SWI-PROLOG корректны. Указатель “курсор” можно использовать для выверки (например, корректности) расстановки скобок. Зелёным цветом выделяются комментарии, темно-красным цветом - переменные. Подчеркиванием выделяются предикаты в теле правила, которые совпадают с предикатом заголовка,- таким образом акцентируется внимание на возможном зацикливании программы.

Чтобы запустить программу, сначала необходимо ее загрузить в SWI-PROLOG для выполнения. Это делается выбором опции Compile/Compile buffer из окна редактора. Результат компиляции отображается в окне интерпретатора SWI-PROLOG. Там же указываются ошибки, возникшие при компиляции, чаще всего они отображаются и во всплывающем окне ошибок. Обычно перед компиляцией предлагается сохранить файл.

Другой способ загрузить уже существующий файл – это выполнение команды Consult в подменю File диалогового окна SWI-PROLOG. На экране появится диалоговое окно.

Рисунок 1.3 Диалоговое окно

Укажите имя файла, который вы хотите загрузить, и выберите Открыть. Если вы попытаетесь загрузить для выполнения файл, в котором есть синтаксические ошибки, то он не загрузится, а вы получите сообщение об ошибке в главном окне. Угловые скобки << >> будут выделять место, где встретилась ошибка. По умолчанию файлы, ассоциируемые с SWI-PROLOG имеют расширение.pl.

Файлы также можно загрузить, используя встроенный предикат:

```
consult(Имя файла или имена нескольких файлов).
```

Листинг 1.2

```
consult(Test).   % test – имя файла
consult([Test1,Test2]).    % Загрузка двух файлов.
consult('test.pl').1
```

Для выполнения загрузки этот предикат нужно написать в главном окне после приглашения интерпретатора (?-), которое означает, что интерпретатор ждет запрос.

Запрос

Запрос это конструкция вида:

?- P1,P2,…,Pn.

которая читается "Верно ли P1 и P2 и ... Pn ?". Предикаты Pi называются подцелями запроса.

Запрос является способом запуска механизма логического вывода, т.е фактически запускает Пролог-программу.

Для просмотра предложений загруженной базы знаний можно использовать встроенный предикат listing.

Проверьте загрузку исходного файла (Листинг 1.1), задайте запрос:

```
?-listing.
```

Введите запрос:

```
?-любит('Билл',бейсбол).   % Любит ли Билл бейсбол?
```

Получите ответ yes (да) и новое приглашение к запросу. Введите следующие запросы и посмотрите на результаты.

Листинг 1.3

```
?-любит('Билл', теннис).   %Любит ли Билл теннис?
```

```
?-любит(Кто, теннис). %Кто любит теннис?

?-любит('Марк',Что),любит('Эллен',Что).%Что любят Марк и
Эллен?

?-любит(Кто, Что).    %Кто что любит?

?-любит(Кто, _). %Кто любит?
```

При поиске решений в базе Пролога выдается первое решение.

Листинг 1.4

```
?-любит(Кто,теннис).

Кто = 'Эллен'
```

Если необходимо продолжить поиск в базе по этому же запросу и получить альтернативные решения, то вводится точка с запятой;

Если необходимо прервать выполнение запроса, (например, нужно набрать другой запрос), используйте клавишу b.

Не вводите имя файла с расширением без апострофов.

Если Вы хотите повторить один из предыдущих запросов, воспользуйтесь клавишами "стрелка вверх" или "стрелка вниз".

Перезагрузить, измененные во внешнем редакторе, файлы можно, ис- пользуя встроенный предикат make. Например так: ?-make.

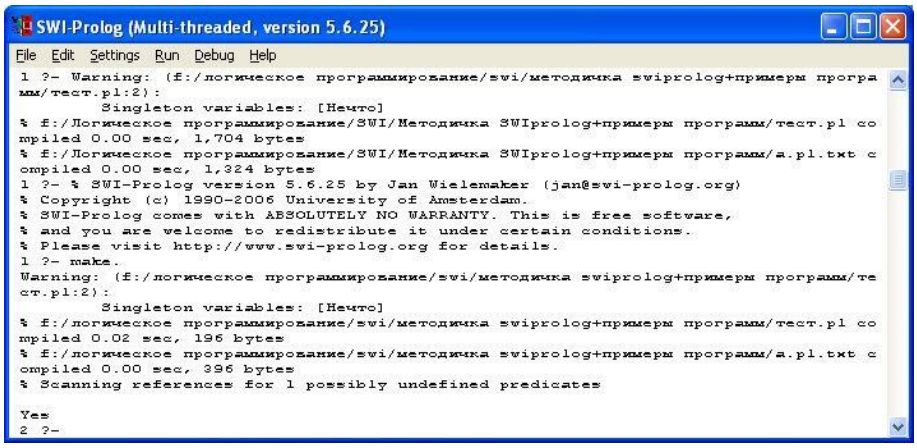

Рисунок 1.4 – встроенный предикат make

Перезагружаются все измененные файлы и файл начальной инициализации pl.ini; о его назначении будет оговорено позднее.

Использование встроенных предикатов

В интерпретаторе SWI-PROLOG имеются, как было отмечено выше, множество встроенных предикатов. Опишем некоторые из них:

consult (Имя файла).

– загрузка одного или нескольких файлов.

```
?- consult(test). test – имя файла
```

pwd

– показывает текущий рабочий каталог.

```
?- pwd. c:/prolog_workspace
```

make

– перезагрузить все измененные файлы (*.pl). В том числе и pl.ini. Аналог

consult. Удобна при редактировании файлов во внешнем редакторе.

```
?- make.
```

working_directory(X,Y)

смена рабочего каталога, где X – текущий рабочий каталог, Y – новый ра- бочий каталог.

ls

просмотр списка файлов в текущем рабочем каталоге.

```
?- ls. a.pl
```

Задание 1.2

Переведите предложения русского языка в предикатную форму, создайте, сохраните и загрузите базу знаний.

Эллен любит чтение. Марк любит компьютеры. Джон любит бадминтон. Эрик любит чтение.

Бадминтон - это вид спорта. Теннис - это вид спорта.

Футбол - это вид спорта. Бейсбол - это вид спорта

Спортсмен - это тот, кто любит какой-нибудь вид спорта.

Объедините эту базу с базой из Задания 1.1.

Выполнение и трассировка программы

Алгоритм выполнения Пролог-программы основан на механизме прямого перебора с возвратом и операции сопоставления (унификации).

Выполнение программы начинается с запроса. Пролог-система берет первую подцель запроса и пытается доказать ее истинность. Для этого просматриваются программа (сверху вниз) и ищется первое утверждение, функтор и арность которого совпадают с функтором и арностью этой подцели.

Если такое утверждение существует и происходит успешное сопоставление аргументов, тогда в случае утверждения-факта - подцель доказана, и Пролог-система переходит к доказательству следующей подцели. В случае утверждения-правила Пролог-система пытается доказать истинность подцелей тела правила слева направо.

Если при поиске решения обнаружено несколько вариантов доказательства истинности цели, то Пролог-система запоминает альтернативные варианты решения - точки возврата.

Если для некоторой цели нет ни одного утверждения, с которым ее можно сопоставить, то цель считается неуспешной. Если при этом остались непройденные точки возврата, то выполняется возврат (бектрекинг) и еще раз делается попытка доказательства подцели.

Выполнение программы на языке Пролог

Листинг 1.5

```
База знаний:
?- a(X),b(X),e. a(1).
a(Y):- c(Y),d(Y). b(2).
c(1).
c(2).
d(2).
e.
```

```
Запрос
?- a(X),b(X),e. ДЕРЕВО ВЫВОДА
```

На рисунках представлено дерево вывода. Жирными линиями в дереве обозначены И-деревья – для доказательства такого дерева необходимо, чтобы каждая его ветка «опиралась» на истинное утверждение. ИЛИ-деревья исходят из вершин с точками возврата, такое дерево истинно, если хотя бы одна ветка опирается на истинное утверждение. Пунктирные линии – альтернативные ветки, не рассматриваемые на данном этапе (либо ложные, либо еще не рассмотренные).

В данном примере получение решения складывается из трех этапов. Вначале рассматривается первая подцель – она имеет два предложения в базе, заголовки которых сопоставимы с ней, поэтому она является точкой возврата. Выбирается первое сверху предложение – это факт a(1), при этом свободная переменная X получает значение 1. Первая цель запроса доказана. Вторая цель - b(1) (X уже связана со значением 1) не сопоставляется ни с одним правилом базы, следовательно, является ложной.

Бэктрекинг – возврат к ближайшей точке возврата, т.е. a(X). На втором дереве отображен вывод по второй альтернативе. Связываются две переменные X из запроса и Y из правила для предиката a. Фактически запрос ?-a(X),b(X),e подменяется на новый набор целей ?-c(Y),d(Y),b(Y),e. Первая подцель – это c(Y) – она является новой точкой ветвления, так как унифицируется с двумя предложениями в базе. Первое предложение – факт приводит к унификации Y с 1, но следующая в наборе подцель d(1) – ложна.

Третье дерево отражает альтернативу для c(Y) с унификацией Y=2. Для этой альтернативы истинны все последующие цели из набора (они унифицируются с соответствующими фактами).

Задание 1.3

Загрузите Пролог-программу из задания 1.2.

В SWI-PROLOG есть два вида отладчиков: командный2 и графический3, в последнем трассировка выглядит более наглядно.

Для перехода в режим трассировки необходимо набрать встроенный предикат debug. Для выхода из режима трассировки – предикат nodebug4.

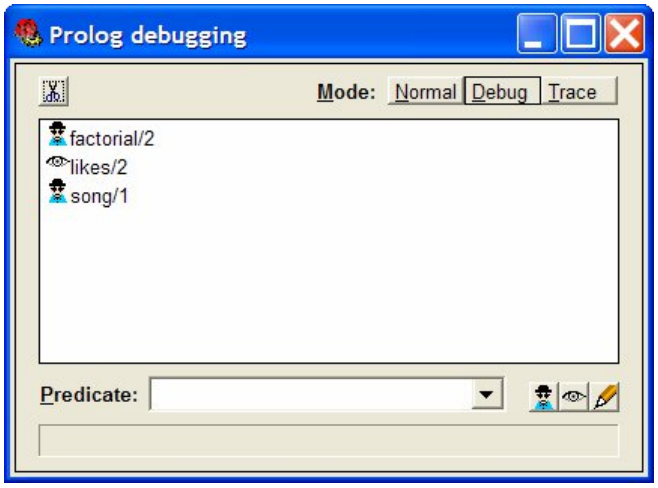

Рисунок 1.5 – Режим трассировки

Перейдите в режим трассировки. Для трассировки необходимо указать контрольные точки (Spy points) в меню Debug->Edit spy points->Predicate, где указать имена предикатов. В данном случае укажите предикат любит. Обрати- те внимание, что справа внизу находятся три кнопки, где указываются как кон- трольные точки, так и точки трассировки. Контрольные точки (Spy points) позволяют вызвать графический отладчик.

Точки трассировки (Trace points)— консольный отладчик. Третий выбор — просмотр и редактирование файла программы, содержащей, введенный в поле Predicate, предикат. Внешний вид изображен на снимке ниже.

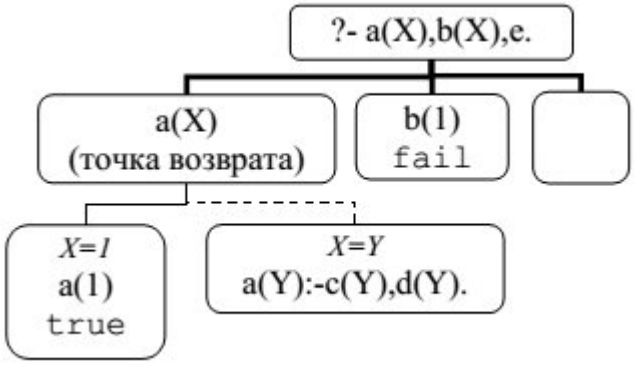

Рисунок 1.6 – Выполнение программы

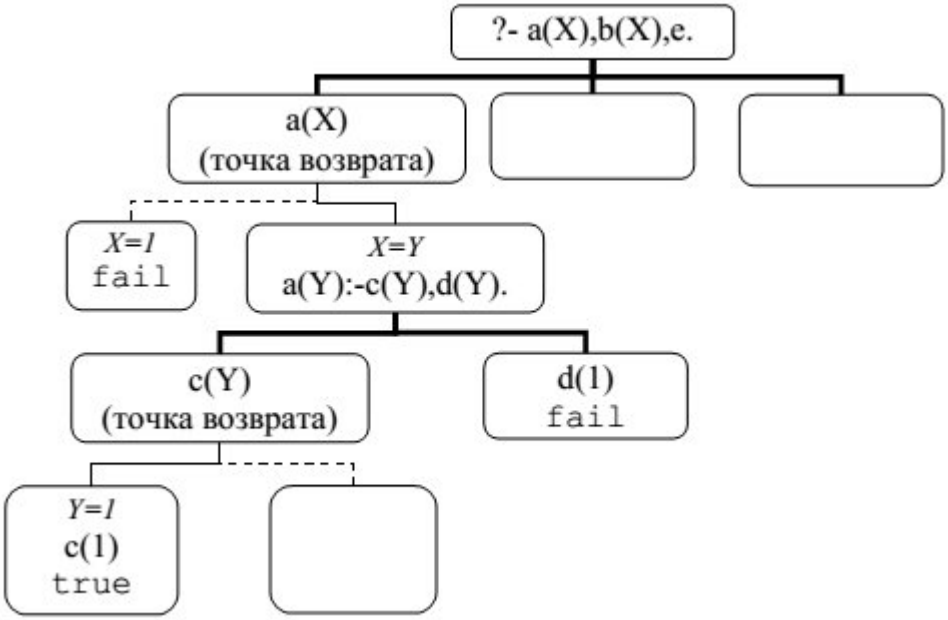

Рисунок 1.7 – Выполнение программы

Графический отладчик не поддерживал русских предикатов (не работает корректно указание контрольных точек и точек трассировки через графический интерфейс). В связи с этим опишем предикаты, относящиеся к отладке программы с использованием командной строки.

Таблица 1.1 – Предикаты трассировки

Предикат	Описание
trace	Включить режим трассировки. Если не указаны точки трассировки, будет производиться полная пошаговая трассировка.
tracing	Успешен, если вызывается из режима трассировки.
notrace	Выход из режима трассировки.

guitracer, gtrace	Включить графического режима трассировки. Окно графического режима трассировки включается при встрече первой контрольной точки (spy-point).
noguitracer	Отключить графического режима трассировки.
trace(Pred)	Установить точку трассировки предиката с именем Pred (то есть отображаться при трассировке будет каждое событие связанное с предикатом Pred).
trace(Pred,Ports)	Установить точку трассировки предиката с именем Pred и активирующейся по событиям (при прохож- дении по портам) Ports. События могут быть следующих типов: fail – событие соответствующее неудачному вызову предиката с именем Pred; call – событие соответствующее первому вы- зову предиката с именем Pred; redo – событие соответствующее повторному вызову предиката с именем Pred; exit – событие соответствующее успешному окончанию вызова предиката с именем Pred. Сам же параметр Ports может принимать значения типов сообщений с префиксами соответствующими добавлению и удалению события (порта) из точки трассировки а так же списки таких значений. Например, запрос ?- trace(foo/2, +fail), trace(foo/2, [+call,-fail]). добавляет в точку трассировки связанную с преди- катом foo/2 событие fail. Для добавления события используют префикс +, для изъятия -. Кроме того можно оперировать сразу со всеми типами событий, используя обозначение all. Пример: `trace(foo/2, +all).`
debug	Включить режим отладки. При запросах к программе в режиме отладки выводятся все события, происходящие при выполнении запроса, связанные с установленными точками отладки.
nodebug	Отключить режим отладки.
debugging	Показать отслеживающиеся точки трассировки и контрольные точки
spy(Pred)	Установить контрольную точку, связанную с предикатом Pred.
ospy(Pred)	Убрать контрольную точку, связанную с предикатом Pred.
nospyall	Убрать все контрольные точки.

2 Отладка происходит в командной строке в диалоговом окне SWI-PROLOG и отображает последовательность выполняемых подцелей и результат выполнения.

3 Графический отладчик отображает информацию в собственном – отдельном окне.

4 Не забывайте каждый раз ставить точку после предикатов или группы предикатов (записанных через запятую) – каждое предложение заканчивается точкой.

Тогда именно на указанных предикатах трассировка будет останавливаться и показывать промежуточный результат. Остальные не указанные предикаты не будут показаны (если не указаны никакие контрольные точки, то режим трассировки будет неотличим от обычного режима работы интерпретатора).

Задайте запрос

```
?-любит(X,теннис),любит(X,компьютеры).
```

и посмотрите, что произойдет. После получения первого решения получите все остальные, вводя;. Трассировка изображена на следующем рисунке:

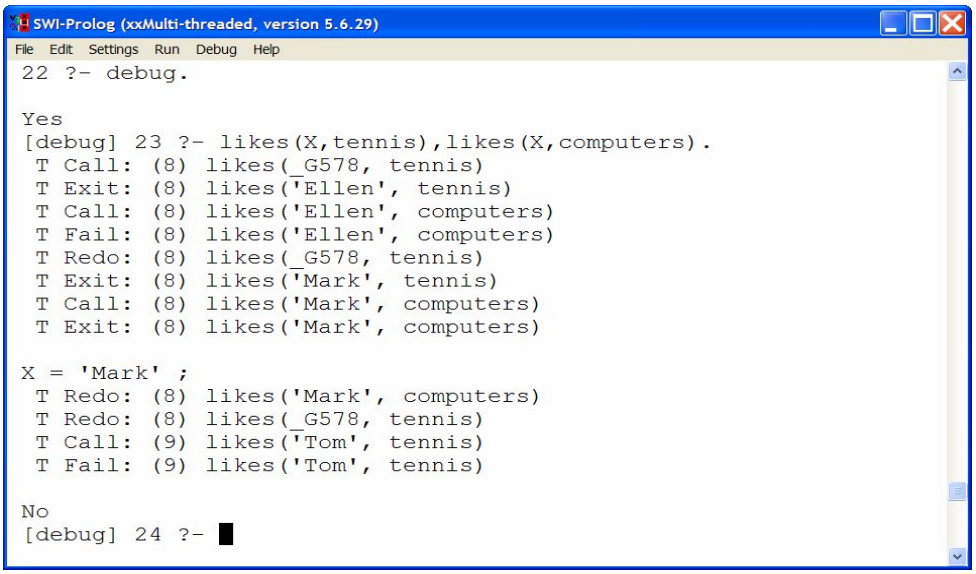

Рисунок 1.7 – Трассировка программы

В левой верхней части окна графического отладчика отображены текущие значения переменных текущего выполняемого предиката, справа стек вызовов предикатов, в нижней части факты и правила, где цветом отмечен следующий вызываемый предикат. Внешний вид графического отладчика изображен на рисунке. Сверху, в виде кнопок, находятся допустимые действия. Вторая кнопка (стрелка вправо) позволяет выполнять программу пошагово (предикат за предикатом). Цвет выделения также имеет значение. Красный цвет обозначает неудачу, ложность решения.

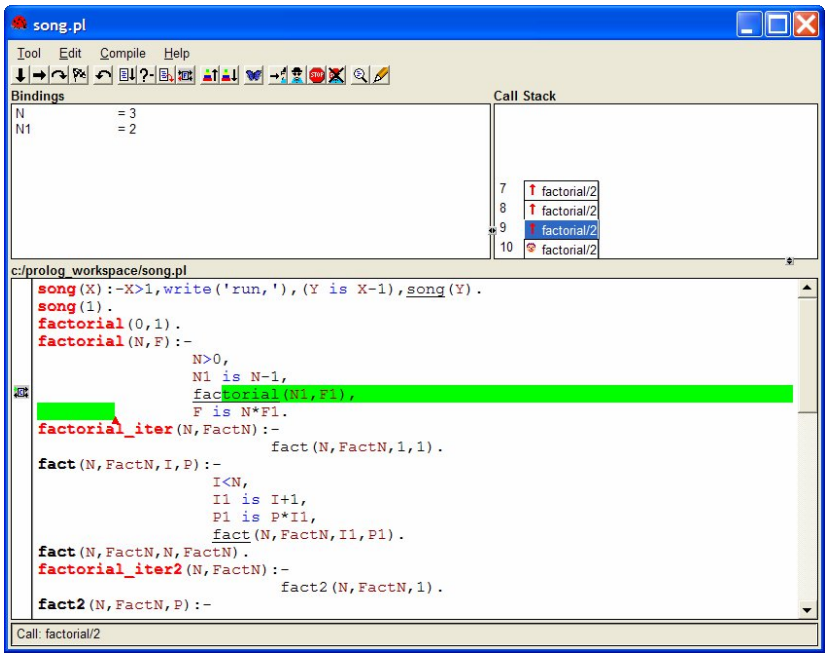

Рисунок 1.8 – Результат трассировки

Постановка задачи к лабораторной работе 1

1. Изучить предлагаемый теоретический материал.

2. Разработайте программы и сформируйте к ним запросы, используя фрагменты кода из листингов 1.1-1.5.

3. Выполните задания 1.1-1.5.

4. Отчет должен содержать титульный лист, включающий название лабораторной работы и номер варианта, а также сведения об авторе (фамилия студента и номер группы).

5. Оформление каждого задания начинайте с указания его номера, например «Задание № 1» и текста задания.

6. Оформляя решение заданий, следует включать в документ скриншоты и комментарии для пояснения используемых обозначений и выполняемых действий.

7. Оформить отчет по лабораторной работе. Представить отчет по лабораторной работе для защиты.

Задание 1.1

Для того чтобы набрать текст программы, воспользуйтесь встроенным текстовым редактором. Чтобы создать новый файл выберете команду File/New, в диалоговом окне укажите имя нового файла, например, тест. Для редактирования уже созданного файла с использованием встроенного редактора можно воспользоваться командой меню File/Edit. Набейте программу 1 (текст программы выше по тексту) и сохраните ее (File/Save buffer).

Задание 1.2

Переведите предложения русского языка в предикатную форму, создайте, сохраните и загрузите базу знаний.

Эллен любит чтение. Марк любит компьютеры. Джон любит бадминтон. Эрик любит чтение.

Бадминтон - это вид спорта. Теннис - это вид спорта.

Футбол - это вид спорта. Бейсбол - это вид спорта

Спортсмен - это тот, кто любит какой-нибудь вид спорта.

Объедините эту базу с базой из Задания 1.1.

Задание 1.3

Загрузите Пролог-программу из задания 1.2.

Задание 1.4

Используя команды внешнего редактора, заменить всюду 'Eric' на 'Tom'. Построить дерево вывода для построенной в задании 1.2 базы данных и запроса

?-sportsman('Mark').

Задание 1.5

Построить базу знаний на основе текста:

Живет зебра на земле. Живет собака на земле. Живет карп в воде. Жи- вет кит в воде. Кошка живет там же, где живет собака. Живет крокодил в воде и на земле. Живет лягушка в воде и на земле. Живет утка в воде, на земле и в воздухе. Живет орел в воздухе и на земле. Живет буревестник в воде и в воздухе.

Задайте к этой базе запросы (воспользуйтесь предикатом help для поис- ка отношения "неравно").

Кто где живет?

Кто живет на земле, но не является собакой? Кто живет и на земле, и в воде?

Варианты индивидуальных заданий

Разработайте техническое задание на проектирование экспертной системы в соответствии с вариантом. Обратите внимание на функциональность и состав запросов к экспертной системе, а также на состав базы знаний проектируемой системы.

Таблица 1.1 –Варианты заданий

№	Предметная область	Целевая аудитория

1	Бухгалтерия	Руководитель предприятия
2	Юриспруденция	Руководитель предприятия
3	HR-служба	Соискатель
4	Компьютерный магазин	Покупатель
5	Магазин электроники	Покупатель
6	Книжный магазин	Менеджер по продажам
7	Агентство недвижимости	Менеджер по продажам
8	Строительная фирма	Покупатель
9	Турфирма	Менеджер по продажам
10	КМВ как туристский кластер	Клиент

Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

Отчет по лабораторной работе должен состоять из следующих структурных элементов:

титульный лист;
вводная часть;
основная часть: описание работы, включающее скриншоты действий;
заключение (выводы).

Вводная часть отчета должна включать пункты:

условие задачи;
порядок выполнения.
программно-аппаратные средства, используемые при выполнении работы.

Зашита отчета по лабораторной работе заключается в предъявлении преподавателю полученных результатов в виде файла и демонстрации полученных навыков при ответах на вопросы преподавателя.

Контрольные вопросы

1. Как можно загрузить программу из файла?
2. Как запускается программа на Прологе?
3. Как отображаются в программе на Прологе свойства сущностей?
4. Как отображаются в программе на Прологе отношения между сущностями?
5. Как отобразить в программе на Прологе проверку конъюнкции двух предикатов?
6. Что такое унификация в Прологе?
7. Что такое правило в Прологе?
8. Опишите функциональные возможности логического программирования.
9. Какими возможностями обладает база знаний?
10. Что такое логическое и функциональное программирование?

Лабораторная работа 2. Структура программы на языке PROLOG. Синтаксис языка PROLOG

Цель работы: Изучение структуры программы на языке PROLOG. Изучение синтаксиса языка PROLOG.

Основы теории

Синтаксис языка Пролог

При формальном описании синтаксиса конструкций алгоритмических языков часто используется так называемая «нормальная форма Бэкуса-Наура» (БНФ), разработанная в 1960 Джоном Бэкусом и Питером Науром. Впервые БНФ была применена Питером Науэром при записи синтаксиса языка Алгол-60. Основными конструкциями БНФ являются следующие.

Символ «::=» читается как «по определению» («это», «есть»). Слева от разделителя располагается объясняемое понятие, справа - конструкция, разъясняющая его. Например,

<Имя> ::= <Идентификатор>

В угловые скобки «< >» заключается часть выражения, которая используется для обозначения синтаксической конструкции языка, в частности объясняемое понятие. В приведенном выше примере это <Имя> и <Идентификатор>.

Символ «|» означает в нотации БНФ «или». Он применяется для разделения альтернативных толкований определяемого понятия. Например, десятичную цифру можно определить следующим образом:

<цифра> ::= 0|1|2|3|4|5|6|7|8|9

Часть синтаксической конструкции, заключенная в квадратные скобки, является необязательной (может присутствовать или отсутствовать), например

<Целое число> ::= [-]<Положительное целое число>

означает, что целое число можно определить через положительное целое число, перед которым может стоять знак минус.

Символ «*» обозначает, что часть синтаксической конструкции может повторяться произвольное число раз (ноль и более). Заметим, что иногда вместо символа «*» используют фигурные скобки «{ }». Например, положительное целое число в нотации БНФ можно следующим образом:

<Положительное целое число> ::= <цифра>[<цифра>]*

или

<Положительное целое число> ::= <цифра>{<цифра>}.

Т.е. положительное целое число состоит из одной или нескольких цифр.

Программа на Прологе состоит из предложений (утверждений). Каждое предложение заканчивается точкой.

Предложения бывают трех видов: факты, правила, вопросы.

Предложение имеет вид

B :-

A1, ... , An.

B называется заголовком или головой предложения, а A1, ..., An - телом. Предложение является перевернутой импликацией (B :- A эквивалентно B ¬ A, «B следует из A») или фразой Хорна. Символ «:-» означает «следует из», символ «,» – конъюнкция (логическое И, Ù).

При необходимости применения дизъюнкции (логическое ИЛИ, Ú) используется символ «;», действующий до следующей дизъюнкции, окончания правила или закрывающей круглой скобки. Например,

Пролог	Логика предикатов
C :- A1, ... , An; B1, ... , Bm.	(A1 Ù ... Ù An) Ú (B1 Ù ... Ù Bm) ® C
или	
D :- A, (B ; C).	A Ù (B Ú C) ® D

Факт фиксирует (определяет) некоторое отношение между объектами. Он состоит только из заголовка. Можно считать, что факт – это предложение, у которого нет тела. В терминах логики предикатов, факт – это и есть предикат.

Например, факт, что Наташа является мамой Даши, может быть записан в виде (в SWI-Prolog строки-константы записываются в одинарных кавычках):

mama('Наташа', 'Даша').

Факт представляет собой безусловно истинное утверждение.

Если воспользоваться нормальной формой Бэкуса-Науэра, то предикат можно определить следующим образом:

<Предикат> ::= <Имя> | <Имя>(<аргумент> [,<аргумент>]*),

т.е. предикат состоит либо только из имени, либо из имени и следующей за ним последовательности аргументов (термов), заключенной в скобки.

Аргументом (термом) предиката может быть константа, переменная или составной объект (список или функция). Число аргументов предиката называется арностью.

Правило – предложение, истинность которого зависит от истинности одного или нескольких предложений. Обычно правило содержит несколько хвостовых целей, которые должны быть истинными для того, чтобы само правило было истинным.

В нотации БНФ правило будет иметь вид:

<Правило> ::= <предикат> :- <предикат> [,<предикат>]*.

Например. Известно, что бабушка человека - это мама его мамы или мама его папы. Соответствующие правила будут иметь вид:

```
grandmama(X, Y):-
mama(X, Z), mama(Z, Y).
grandmama(X, Y):-
mama(X, Z), papa(Z, Y).
или
grandmama(X, Y):-
mama(X, Z), mama(Z, Y);
mama(X, Z), papa(Z, Y).
или
grandmama(X, Y):-
mama(X, Z), (mama(Z, Y); papa(Z, Y)).
```

Имя переменной в Прологе может состоять из букв латинского алфавита, цифр, знаков подчеркивания и должно начинаться с прописной буквы или знака подчеркивания. При этом переменные в теле правила неявно связаны квантором всеобщности и эквивалентны объектам предметной области.

Переменные могут быть свободными или связанными.

Свободная переменная – переменная, которая еще не получила значения. Она не равняется ни нулю, ни пробелу; у нее вообще нет никакого значения. Такие переменные еще называют неконкретизированными.

Переменная, которая получила какое-то значение и оказалась связанной с определенным объектом, называется связанной. Если переменная была конкретизирована каким-то значением и ей сопоставлен некоторый объект, то эта переменная уже не может быть изменена.

Областью действия переменной в Прологе является одно предложение. В разных предложениях может использоваться одно и то же имя переменной для обозначения разных объектов. Исключением из правила определения области действия является анонимная переменная, которая обозначается символом подчеркивания «_». Анонимная переменная предписывает интерпретатору (компилятору) проигнорировать значение аргумента (терма). Если в правиле несколько анонимных переменных, то все они отличаются друг от друга, несмотря на то, что записаны с использованием одного и того же символа («_»). Анонимные переменные могут записываться только в качестве терма предиката. Использовать их в выражениях (например, арифметических) нельзя.

Третьим специфическим видом предложений Пролога можно считать вопросы (запросы, цели).

Вопрос состоит только из тела и может быть выражен с помощью БНФ в виде:

<Вопрос> ::= <Предикат> [,<Предикат>]*

Вопросы используют для выяснения выполнимости некоторого отношения между описанными в программе объектами. Система рассматривает вопрос как цель, к которой надо стремиться. Ответ на вопрос может оказаться положительным (true) или отрицательным (false), в зависимости от того, может ли быть достигнута соответствующая цель.

Программа может содержать вопрос в теле (внутренняя цель). Если программа содержит внутреннюю цель, то после запуска программы на выполнение система сразу проверяет достижимость заданной цели. Если внутренней цели в программе нет, то после запуска программы система выдает приглашение вводить вопросы в диалоговом режиме (внешняя цель). Программа, компилируемая в исполняемый файл, обязательно должна иметь внутреннюю цель.

Если цель достигнута, система отвечает «yes» («true»), в противном случае «no» («false»). Следует отметить, что ответ «no» на вопрос не всегда означает, что он отрицательный. Система может дать такой ответ и в том случае, когда у нее просто недостаточно информации, позволяющей положительно ответить на вопрос. Т.е. Пролог основан на т.н. «модели закрытого мира», в которой все, что можно получить на основе описания модели является истиной, а остальное – ложью.

Факт, правило и вопрос с точки зрения фраз Хорна (B ¬ A) можно интерпретировать следующим образом:

- факт: B ¬ true;
- правило: B ¬ A;
- вопрос: true ¬ A.

Т.о. программа (база знаний) на Прологе состоит из фактов и правил, которые представляют собой продукции с предикатами в левой и правой части. Тем самым программа выражает некоторые знания о предметной области, необходимые для решения задачи. Запрос - это также некоторый предикат, истинность которого нас интересует. Если запрос не содержит переменных, то вычисление его значения дает ответ «true» при его истинности, либо ответ «false» при его ложности. Если же в предикате запроса есть переменные, то ищутся их значения (интерпретация), при которых этот предикат и все предикаты программы становятся истинными. В этом и состоит суть вычислений (логического вывода) программы на Прологе.

Пример программирования на языке Пролог

Рассмотрим несколько примеров. Пусть в программе заданы следующие факты (предикаты) и правила:

Листинг 2.1

```
mama('Наташа', 'Даша').
mama('Даша', 'Маша').
mama('Наташа', 'Вася').

papa('Вася', 'Маша').

grandmama(X, Y):-
mama(X, Z), (mama(Z, Y); papa(Z, Y)).

grandpapa(X, Y):-
papa(X, Z), (mama(Z, Y); papa(Z, Y)).
```

Вопрос 1 – является ли Наташа мамой Даши:

```
?- mama('Наташа', 'Даша').
true.
```

Вопрос 2 – кто является мамой Даши:

```
?- mama(X, 'Даша').
X = 'Наташа' ;
false.
```

Первая строка сообщения означает, что ответ найдет и мамой Даши является Наташа; вторая – что в базе знаний для оставшихся предложений не обнаружены другие мамы Даши.

Вопрос 3 – есть ли у Даши мама:

```
?- mama(_, 'Даша').
true.
```

Вопрос 4 – найти всех мам и детей:

```
?- mama(X,Y).
X = 'Наташа',
Y = 'Даша' ;
X = 'Даша',
Y = 'Маша' ;
X = 'Наташа',
Y = 'Вася'.
```

В данном случае после третьего ответа не выдается «false», т.к. в базе знаний были перебраны все предложения и они все истинны.

Вопрос 5 – для кого Наташа является бабушкой:

```
?- grandmama('Наташа', X).
X = 'Маша' ;
X = 'Маша'.
```

В данном случае выдается два одинаковых ответа «Маша», т.к. правило grandmama в одном случае сработало по цепочке «Наташа – Даша – Маша», а в другом - «Наташа – Вася – Маша». Очевидно, что в приведенном примере базы знаний либо Наташи - это две различные женщины, либо Маши.

Краткие сведения об операциях и встроенных предикатах SWI-Prolog

В табл. 2.1 приведены некоторые операции и предикаты SWI-Prolog, которые в дальнейшем будут использоваться для иллюстрации примеров.

Таблица 2.1 Некоторые операции и предикаты SWI-Prolog

Операция / Предикат	Назначение
true	Истина
fail, false	Ложь
=	Унификация (присваивание значения несвязанной переменной)

<, =<, >=, >	Арифметические (только для чисел) операции сравнения
=:=	Арифметическое равенство
=\=	Арифметическое неравенство
is	Вычисление арифметического выражения (например, A is 5 + 2)
@<, @=<, @>=, @>	Операции сравнения для констант и переменных любого типа (чисел, строк, списков и т.д.)
==	Равенство констант и переменных любого типа
\==	Неравенство констант и переменных любого типа
not(A)	Отрицание логического выражения A
read(A)	Чтение значения с клавиатуры и присваивание его переменной A
write(A)	Печать A на экран с установкой курсора после последнего напечатанного символа
writeln(A)	Печать A на экран с переводом курсора в начало следующей строки
nl	Перевод курсора в начало следующей строки
repeat	Предикат, выдающий новое истинное значение при возврате. Передоказываемый предикат
!	Предикат (cut, сократить), запрещающий возврат далее той точки, где он стоит
assert(A), assertz(A)	Динамическое добавление факта (правила) в конец списка подобных фактов (правил) базы знаний (программы)
asserta(A)	Динамическое добавление факта (правила) в начало списка подобных фактов (правил) базы знаний
retract(A)	Удаление первого факта (правила) базы знаний
retractall(A)	Удаление всех фактов (правил) базы знаний с именем A

Процедура вывода в Прологе

При поиске решения (доказательства цели) в Прологе используется метод перебора с возвратами (поиск в глубину). Пролог при доказательстве утверждения поочередно пытается установить истинность входящих в него предикатов (утверждений). Если первый предикат истинен, то Пролог переходит ко второму. Если и он истинен, то переходит к третьему. Если второй предикат ложен, то Пролог пытается установить его истинность при других значениях, входящих в него переменных. Если этого не удается сделать, то он возвращается к первому предикату и пытается установить его истинность для новых значений переменных, а затем снова возвращается к доказательству второго предиката. Такая процедура повторяется до тех пор, пока не будет достигнута истинность последнего предиката. После доказательства истинности последнего предиката цели Пролог завершает работу. Процесс возврата в Прологе называется backtracking.

Например, пусть имеется запрос на определения внучек и внуков «Наташи»:

```
?- mama('Наташа', Y),
(mama(Y, Z); papa(Y, Z)),
write(Z);
write('Всё').
Ответ.
Маша;
Маша;
Всё;
```

Выделенная жирным часть соответствует правилу определения бабушки (grandmama).

Постановка задачи к лабораторной работе 2

1. Изучить предлагаемый теоретический материал.
2. Отчет должен содержать титульный лист, включающий название лабораторной работы и номер варианта, а также сведения об авторе (фамилия студента и номер группы).
3. Оформление каждого задания начинайте с указания его номера, например «Задание № 1» и текста задания.
4. Оформляя решение заданий, следует включать в документ скриншоты и комментарии для пояснения используемых обозначений и выполняемых действий.
5. Оформить отчет по лабораторной работе. Представить отчет по лабораторной работе для защиты.

Задания к лабораторной работе

В соответствии с полученным заданием сформировать решение и дать его описание со скриншотами выполненных действий.

Покажите на скриншотах результат запросов, приведенных ниже, используя к качестве базовой программы Листинг 2.1.

Вопрос 1 – является ли Наташа мамой Даши:

```
?- mama('Наташа', 'Даша').
true.
```

Вопрос 2 – кто является мамой Даши:

```
?- mama(X, 'Даша').
X = 'Наташа' ;
false.
```

Первая строка сообщения означает, что ответ найдет и мамой Даши является Наташа; вторая – что в базе знаний для оставшихся предложений не обнаружены другие мамы Даши.

Вопрос 3 – есть ли у Даши мама:

```
?- mama(_, 'Даша').
true.
```

Вопрос 4 – найти всех мам и детей:

```
?- mama(X,Y).
X = 'Наташа',
Y = 'Даша' ;
X = 'Даша',
Y = 'Маша' ;
X = 'Наташа',
Y = 'Вася'.
```

В данном случае после третьего ответа не выдается «false», т.к. в базе знаний были перебраны все предложения и они все истинны.

Вопрос 5 – для кого Наташа является бабушкой:

```
?- grandmama('Наташа', X).
X = 'Маша' ;
X = 'Маша'.
```

В данном случае выдается два одинаковых ответа «Маша», т.к. правило grandmama в одном случае сработало по цепочке «Наташа – Даша – Маша», а в другом - «Наташа – Вася – Маша». Очевидно, что в приведенном примере базы знаний либо Наташи - это две различные женщины, либо Маши.

Варианты индивидуальных заданий

В соответствии с полученным заданием сформировать решение и дать его описание со скриншотами выполненных действий.

1. Покажите на скриншотах результат запросов, приведенных ниже, используя к качестве базовой программы Листинг 2.1. Выполните операции, указанные в таблице 2.1

Таблица 2.3 – Выполнение простейших операций

№ п/п	Предикат запроса	Проверяемый предикат базы знаний	Результат
1	**mama('Наташа', Y)**	**mama('Наташа', 'Даша')**	**Y = 'Даша'**
2	mama(Y, Z) ° mama('Даша', Z)	mama('Наташа', 'Даша')	backtracking
3	**mama(Y, Z) ° mama('Даша', Z)**	**mama('Даша', 'Маша')**	**Z = 'Маша'**
4	**write(Z) ° write('Маша')**		**'Маша'**
5	mama(Y, Z) ° mama('Даша', Z)	mama('Наташа', 'Вася')	backtracking
6	papa(Y, Z) ° papa('Даша', Z)	papa('Вася', 'Маша')	backtracking
7	mama('Наташа', Y)	mama('Даша', 'Маша')	backtracking
8	**mama('Наташа', Y)**	**mama('Наташа', 'Вася')**	**Y = 'Вася'**
9	mama(Y, Z) ° mama('Вася', Z)	mama('Наташа', 'Даша')	backtracking
10	mama(Y, Z) ° mama('Вася', Z)	mama('Даша', 'Маша')	backtracking
11	mama(Y, Z) ° mama('Вася', Z)	mama('Наташа', 'Вася')	backtracking
12	**papa(Y, Z) ° papa('Вася', Z)**	**papa('Вася', 'Маша')**	**Z = 'Маша'**
13	**write(Z) ° write('Маша')**		**'Маша'**
14	**write('Всё')**		**'Всё'**

1) Указанная последовательность имеет место, если в SWI-Prolog после вывода на консоль имени внучки (внука) нажимать клавишу «;» (поиск альтернативного доказательства).

2) Жирным выделены строки, для которых выполнение предиката истинно.

3) Для используемой базы знаний либо Наташи - это две различные женщины, либо Маши.

Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

Отчет по лабораторной работе должен состоять из следующих структурных элементов:

титульный лист;

вводная часть;
основная часть: описание работы, включающее скриншоты действий;
заключение (выводы).

Вводная часть отчета должна включать пункты:
условие задачи;
порядок выполнения.
программно-аппаратные средства, используемые при выполнении работы.

Зашита отчета по лабораторной работе заключается в предъявлении преподавателю полученных результатов в виде файла и демонстрации полученных навыков при ответах на вопросы преподавателя.

Контрольные вопросы

1. Перечислите достоинства и недостатки логического программирования.
2. Перечислите операторы логического программирования.
3. Для чего применяется трассировка?
4. В каких случаях применяются экспертные системы?
5. Чем отличается логическое программирование от функционального?
6. Дайте определение понятия «предикат»
7. Дайте определение рекурсии
8. В каких случаях является целесообразным использование Пролога?
9. Перечислите основные компоненты интеллектуальных систем.
10. В чем отличие базы данных от базы знаний?

Лабораторная работа 3. Алгоритмы логического программирования. Решение задач на рекурсию

Цель работы: Изучить и применить на практике алгоритмы логического программирования. Выполнить разработку программ, испльзующих рекурсию.

Основы теории

Рекурсия в Прологе

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

Пусть имеются следующие факты о том, какая валюта котируется выше: doroje(dollar, rubl).

Листинг 3.1

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

Выполним запрос:

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

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

```
? doroje(evro, iena).
```

То ответ будет отрицательный, поскольку такой факт отсутствует.

Аналогичным будет ответ на вопрос:

```
? doroje(funt, rubl).
```

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

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

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

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

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

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

Листинг 3.2

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

Первое предложение в этой конструкции определяет момент прекращения рекурсивных вызовов.

Второе правило описывает возможности рекурсивных вызовов, когда существуют непроверенные варианты решения. Вообще, любая рекурсивная процедура должна содержать:

Нерекурсивное правило, применяемое для завершения рекурсии (их может быть несколько, задающие разные условия для завершения рекурсии),

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

Набор правил просматривается сверху вниз. Сначала делается попытка выполнения нерекурсивного правила. Если оно отсутствует, то рекурсивное правило может работать бесконечно.

Применение рекурсии в логическом программировании

Рекурсия – процесс повторения элементов самоподобным образом. **Рекурсия** (в программировании) – алгоритмический метод, заключающийся в возможности обращения правила (функции, процедуры) к самому себе один или более раз.

Рекурсия является часто используемым приемом в программах на Прологе. Для рассмотренного выше примера базы знаний «Родственники» введем правило, описывающее отношение «предок» - «потомок» с помощью рекурсии.

Листинг 3.3 Родственники

```
% 1 параметр - имя отца или матери
% 2 параметр - имя сына или дочери
roditel('Вася', 'Вика').
roditel('Вася', 'Коля').
roditel('Коля', 'Юля').
roditel('Юля', 'Миша').
roditel('Юля', 'Маша').
```

```
roditel('Коля', 'Света').
roditel('Света', 'Рома').
roditel('Света', 'Леша').
% 1 параметр - предок
% 2 параметр - потомок
predok(A, B) :- roditel(A, B).
predok(A, B) :- roditel(A, C), predok(C, B).
```

На вопрос predok('Вася', 'Миша') ответ будет положительным.

Второй пример использования рекурсии – расчет факториала.

Листинг 3.4 Расчет факториала

```
fact(1, 1):- !. % факториал единицы равен единице
fact(N, F):-
N1 is N - 1,
fact(N1, F1), % F1 равен факториалу числа на единицу меньшего N
F is F1 * N. % факториал числа N равен произведению F1 на само число N
```

На вопрос fact(6, F) ответ будет «F = 720».

Любая рекурсивная процедура в Прологе должна включать, как минимум два правила:

1) нерекурсивное правило, определяющее его вид в момент прекращения рекурсии;

2) рекурсивное правило. Первая подцель, вырабатывает новые значения аргументов, а вторая - вызов самого правила с новыми значениями аргументов.

Управление процессом вывода

В Прологе имеется два стандартных предиката, которые позволяют управлять процедурой перебора с возвратами:

- fail (false) – предикат, который всегда возвращает ложь;
- ! – предикат (cut, сократить), запрещающий возврат далее той точки, где он стоит.

Первый из них используется для организации циклов, а второй для ускорения процедуры перебора.

Если мы желаем узнать всех предков «Миши» и при этом, чтобы выдача предков на консоль была выполнена автоматически, а не в диалоговом режиме (путем нажатия «;» после каждого ответа), тогда запрос должен выглядеть следующим образом:

Листинг 3.5

```
?- writeln('Предки Миши:'),
predok(A, 'Миша'),
writeln(A),
fail;
true.
Ответ:
Предки Миши:
```

```
Юля
Вася
Коля
```

Таким образом, предикат fail выполняет принудительный откат и заставляет Пролог заново передоказывать все предыдущие подцели с новыми значениями переменных. Передоказательству в приведенном примере будет подвержен только предикат predok(A, 'Миша'), т.к. предикаты write и writeln не генерируют новые значения переменных. Если после предиката fail находятся другие предикаты правила, указываемые через «,» (логическое И), то они никогда не будут выполнены.

Рассмотрим более сложный пример: выяснить всех правнуков «Коли».

Листинг 3.6

```
?- roditel('Коля', A),
write('Правнуки ('), write(A), writeln('):'),
roditel(A, B),
writeln(B),
fail;
true.
Ответ:
Правнуки (Юля):
Миша
Маша
Правнуки (Света):
Рома
Леша
```

Если в данной цели использовать предикат ! после roditel('Коля', A), то результатом работы программы будет первых три строки, если после roditel(A, B) – первых две строки. Если при выполнении правила в результате возврата для передоказательства предикатов с новыми значениями переменных будет встречен предикат !, то всё правило в целом возвратит «false», даже если у него имеются альтернативные пути доказательства, указываемые через «;» (логическое ИЛИ) или определения, задаваемые отдельными предложениями.

Постановка задачи к лабораторной работе 3

1. Изучить предлагаемый теоретический материал.
2. Разработать программы, используя фрагменты кода (листинги 3.1-3.6).
2. Отчет должен содержать титульный лист, включающий название лабораторной работы и номер варианта, а также сведения об авторе (фамилия студента и номер группы).
3. Оформление каждого задания начинайте с указания его номера, например «Задание № 1» и текста задания.
4. Оформляя решение заданий, следует включать в документ скриншоты и комментарии для пояснения используемых обозначений и выполняемых действий.
5. Оформить отчет по лабораторной работе. Представить отчет по лабораторной работе для защиты.

Варианты индивидуальных заданий

В соответствии с полученным заданием сформировать решение и дать его описание со скриншотами выполненных действий.

Задание 1.

1.1 Составить список фрезерных станков с их параметрами, осуществить поиск станка в данном списке и вывести его параметры. Основной параметр — это ширина стола. Для данных станков ширина стола имеет следующие размеры: 200, 250, 320, 400, 500.

В таблице 1 приведены виды станков с параметрами.

Таблица 3.1 Перечень станков

Тип станка	Название	Размер стола
вертикально-фрезерный	6М10	200 мм
вертикально-фрезерный	6Н11	250 мм
горизонтально-фрезерный	6М80Г	200 мм
горизонтально-фрезерный	6Н81Г	250 мм
вертикально-фрезерный	6Н11	320 мм
горизонтально-фрезерный	6М82Г	320 мм
вертикально-фрезерный	6М13	400 мм
горизонтально-фрезерный	6М83Г	400 мм
вертикально-фрезерный	6Н14	500 мм
горизонтально-фрезерный	6Н84Г	500 мм

1.2 Поэкспериментировать с правилом «принадлежит», изучить с помощью отладчика, как работает рекурсивный перебор элементов списка.

1.3 Создать во встроенном редакторе программу, в которой представлена информация из выше приведенной таблицы в виде унарных предикатов станок с аргументом – списком из трех параметров (из столбцов таблицы), правил логического вывода с рекурсией для поиска параметров станка по заданному одному параметру. Предикатов «станок» должно быть 10 (по одному на каждую строку таблицы). Программа должна содержать следующие правила:

правило найти, с которого запускается программа:

```
найти(X):- станок(L), параметры(X,L), phh(L).
```

где: L – список параметров.

правило «параметры» с рекурсией для поиска заданного параметра X (написать самостоятельно)

правило phh для вывода на экран списка параметров в виде строки (написать самостоятельно).

Задание 2.

Необходимо разработать программу «Родственные связи», отвечающую следующим требованиям.

2.1 Программа должна содержать общие для всех вариантов факты и правила:

- женщина;
- мужчина;
- мать;
- отец;
- родитель;
- супруги.

2.2 По вариантам добавить в программу правила для определения родственников:

1. брат;
2. сестра;
3. ребенок;
4. бабушка;
5. дедушка;
6. внук;
7. внучка;
8. прадедушка;
9. прабабушка;
10. правнук;
11. правнучка;
12. зять (муж дочери, сестры, золовки);
13. невестка (жена сына для его отца, жена брата);
14. свекор (отец мужа);
15. свекровь (мать мужа);
16. тесть (отец жены);
17. теща (мать жены);
18. сноха (жена сына для его матери);
19. сват (отец одного из супругов для родителей другого супруга);
20. сватья (мать одного из супругов для родителей другого супруга);
21. свояк (муж сестры жены);
22. свояченица (сестра жены);
23. свояки (лица, женатые на двух сестрах);
24. деверь (родной брат мужа для жены);
25. золовка (сестра мужа);
26. шурин (брат жены);
27. дядя (брат отца или матери по отношению к детям, племянникам, кроме того, дядей является муж тетки);
28. тетка (сестра отца или матери по отношению к детям, племянникам. кроме того, теткой является жена дяди);
29. племянник (ребенок брата или сестры);
30. внучатый племянник (внук брата или сестры);
31. внучатая племянница (внучка брата или сестры);
32. двоюродный дедушка (дядя отца или матери);
33. двоюродная бабушка (тетка отца или матери);
34. двоюродный брат – кузен (сын родного дяди или тети);
35. двоюродная сестра – кузина (дочь родного дяди или тети);
36. двоюродный дядя (двоюродный брат отца и матери);
37. двоюродная тетка (двоюродная сестра отца и матери);
38. мачеха (неродная мать ребенка);
39. отчим (неродной отец ребенка);
40. падчерица (неродная дочь по отношению к одному из супругов);
41. пасынок (неродной сын по отношению к одному из супругов);
42. сводный брат (сын неродного родителя (мачехи, отчима) по отношению к детям родного);
43. сводная сестра (дочь неродного родителя (мачехи, отчима) по отношению к детям родного);

2.3 При необходимости добавить в программу недостающие факты и правила, требуемые для работы правил из п. 2.2.

Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

Отчет по лабораторной работе должен состоять из следующих структурных элементов:

титульный лист;
вводная часть;
основная часть: описание работы, включающее скриншоты действий;
заключение (выводы).

Вводная часть отчета должна включать пункты:

условие задачи;
порядок выполнения.
программно-аппаратные средства, используемые при выполнении работы.

Зашита отчета по лабораторной работе заключается в предъявлении преподавателю полученных результатов в виде файла и демонстрации полученных навыков при ответах на вопросы преподавателя.

Контрольные вопросы

1. Как можно загрузить программу из файла?
2. Как запускается программа на Прологе?
3. Как отображаются в программе на Прологе свойства сущностей?
4. Как отображаются в программе на Прологе отношения между сущностями?
5. Как отобразить в программе на Прологе проверку конъюнкции двух предикатов?
6. Что такое унификация в Прологе?
7. Что такое правило в Прологе?
8. Опишите функциональные возможности логического программирования
9. Опишите основные свойства рекурсии
10. Как на языке Пролог может быть реализована рекурсия?

Лабораторная работа 4. Применение списков в Прологе

Цель работы: Научиться использовать списки в Прологе.

Основы теории

Использование списков

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

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

Элементами списка могут быть любые термы Пролога, т.е. атомы, числа, переменные и составные термы. Каждый непустой список может быть разделен на голову – первый элемент списка и хвост – остальные элементы списка. При этом список представляется в виде:

```
[A|B] или [1| [2,3]] или [1| B] или [“вчера”|
[“сегодня”,”завтра”]].
```

Это позволяет всякий список представить в виде бинарного дерева (рис.1). У списка, состоящего только из одного элемента, головой является этот элемент, а хвостом – пустой список. Для использования списка необходимо описать предикат списка.

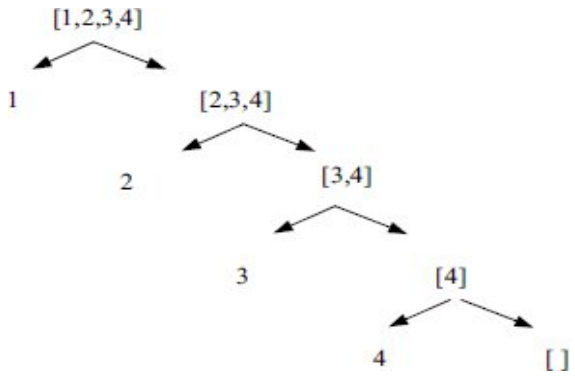

Рисунок 4.1 Бинарное дерево списка.

В следующем примере 1 - голова списка, а [2, 3, 4, 5] - хвост. Пролог покажет это при помощи сопоставления списка чисел с образцом, состоящим из головы и хвоста.

```
?- [1, 2, 3, 4, 5] = [Head | Tail].
Head = 1
Tail = [2, 3, 4, 5]
Yes
```

Здесь Head и Tail - только имена переменных. Мы могли бы использовать X и Y или какие-нибудь другие имена переменных с тем же успехом. Заметим, что хвост списка всегда является списком. Голова, в свою очередь, есть элемент списка, что верно и для всех других элементов, расположенных до вертикальной черты. Это позволяет получить, скажем, второй элемент списка.

В Prolog используется специальный символ для разделения списка на голову и хвост – вертикальная черта |.

Например:

```
[1, 2, 3] или [1 | [2, 3]] или [1 | [2| [3]]] или [1 | [2 | [3 | [ ]]]]
```

Вертикальную черту можно использовать не только для отделения головы списка, но и для отделения произвольного числа начальных элементов списка:

```
[1, 2, 3] или [1, 2 | [3]] или [1, 2, 3 | [ ]]
```

Пример

Используем анонимные переменные для головы и списка, стоящего после черты, если нам нужен только второй элемент списка:

```
?- [слон, лошадь, осел, собака] = [_, X | _ ]. X = лошадь
Yes
```

Рассмотрим несколько процедур обработки списков. Обратите внимание, что все они используют рекурсию, в которой терминальное (базовое) правило определено для пустого списка.

Пример

Предикат «перестановка» выдает списки, полученные перестановкой элементов своего первого аргумента.

```
перестановка([],[]).
перестановка([H|L],Z):- перестановка(L,Y), место(H,Y,Z).
```

Пример использования:

Листинг 4.1

```
?- перестановка([a,b,c],X). X = [a, b, c] ;
X = [b, a, c] ;
X = [b, c, a] ;
X = [a, c, b] ;
X = [c, a, b] ;
X = [c, b, a] ; No
```

Предположим, что имеется некоторый список, в котором X обозначает его голову, а Y — хвост списка. Такой список можно записать, как [X|Y]. Этот список может содержать, например, инструмент для обработки деталей:

[фреза, сверло, резец, хон]

Теперь предположим, что мы хотим определить, содержится ли некоторый инструмент в указанном списке. В Прологе это можно сделать, определив, совпадает ли данный инструмент с головой списка. Если совпадает, то наш список завершается успехом. Если нет, то мы проверяем, есть ли нужный инструмент в хвосте исходного списка. Это значит, что снова проверяется голова, но уже очередного хвоста списка. Если мы доходим до конца списка, который будет пустым списком, то наш поиск завершается неудачей: указанного инструмента в исходном списке нет.

Для того, чтобы записать все это на Прологе, сначала надо:

Установить, что между объектом и списком, в который этот объект может входить, существует отношение. Для записи этого отношения будем использовать предикат принадлежит: целевое утверждение принадлежит(X,Y) является истинным («выполняется»), если терм, связанный с X, является элементом списка, связанного с Y. Имеются два условия, которые надо проверить для определения истинности предиката. Первое условие говорит, что X будет элементом списка Y, если X совпадает с головой списка Y. На Прологе этот факт записывается следующим образом:

```
принадлежит(X,[X|_]).
```

Эта запись констатирует, что X является элементом списка, который имеет X в качестве головы. В данном случае анонимная переменная «_» для обозначения хвоста списка.

Второе правило говорит о том, что X принадлежит списку при условии, что он входит в хвост этого списка, обозначаемый через Y. Для этого используем тот же самый предикат принадлежит для того, чтобы определить, принадлежит ли X хвосту списка. В этом и состоит суть рекурсии. На Прологе это выглядит так:

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

Два этих правила в совокупности определяют предикат для отношения принадлежности и указывают Прологу, каким образом просматривать список от начала до конца при поиске некоторого элемента в списке. Наиболее важный момент, о котором следует помнить, встретившись с рекурсивно определенным предикатом, заключается в том, что прежде всего надо найти граничные условия и способ использования рекурсии.

Для предиката «принадлежит» в действительности имеются два типа граничных условий. Либо объект, который мы ищем, содержится в списке, либо нет. Первое граничное условие для предиката «принадлежит» распознается первым утверждением, которое приводит к прекращению поиска в списке, если первый аргумент предиката принадлежит совпадает с головой списка. Второе граничное условие встречается, когда второй аргумент предиката «принадлежит» является пустым списком.

Это все демонстрирует следующий пример на Прологе: принадлежит(X,[X|_]).

```
принадлежит(X,[_|Y]):- принадлежит(X,Y).
?- принадлежит(фреза,[фреза, сверло, резец, хон]).
true
?- принадлежит(протяжка,[фреза, сверло, резец, хон]).
false.
```

Способы организации циклов

В Прологе отсутствуют конструкции циклов с параметром, пред- и постусловием, но с помощью соответствующих механизмов можно организовать разные типы циклов.

1 способ. Используя рекурсию.

2 способ. Используя предикат, который можно передоказать, и предикат fail.

3 способ. Используя предикат, который можно передоказать, и логическое утверждение. Например, выдать пары «предок» - «потомок» до тех пор, пока не встретится потомок с именем «Миша».

Листинг 4.2

```
?- predok(A, B),
write(A), write(' является предком '), writeln(B),
B == 'Миша';
true.
Ответ:
Вика является предком Коля
Вася является предком Коля
Коля является предком Юля
Юля является предком Миша
```

Такая конструкция соответствует циклу с постусловием.

4 способ. Организация цикла со счетчиком, используя предикат repeat и динамическое добавление фактов в базу знаний (программу).

Следует отметить, что в некоторых реализациях языка Пролог отсутствует встроенный предикат repeat. Тогда данный предикат надо определить в программе следующим образом

repeat.

repeat:- repeat.

Его используют вместо передоказываемого предиката. Но если при передоказательстве обычных предикатов может наступить момент, что все факты исчерпаны (например, predok(A, ‘Миша’) ... fail), то при использовании предиката repeat такой момент не наступает никогда.

Листинг 4.3

```
?- assert(count(1)),
repeat,
retract(count(X)),
X2 is X * X,
write(X), write('^2 = '), writeln(X2),
X1 is X + 1,
```

```
(X1 > 10, !;
assert(count(X1)), fail).
```

Приведенная программа будет циклически выдавать на экран квадраты чисел от 1 до 10. В качестве счетчика используется предикат (факт) count, который динамически добавляется и удаляется из базы знаний.

Существуют также другие способы организации циклов.

Применение списков в Прологе

В Прологе нет такой распространенной и часто используемой структуры хранения данных как массивы, но зато есть развитые возможности работы со списками. Список – упорядоченный набор элементов одного типа. В отличие от массивов, количество элементов которых строго фиксировано (в большинстве языков программирования), списки позволяют модифицировать, добавлять или удалять из него элементы.

Списки в Прологе заключаются в квадратные скобки, например [1, 2, 8, 123] или ['Пн', 'Вт', 'Четверг']. Список, не содержащий ни одного элемента «[]», называется пустым. Каждый непустой список состоит из двух частей: головы и хвоста. Головой является первый элемент списка, хвостом – все остальное.

Таблица 4.1 Списки и его составные части

Список	Голова	Хвост
[1, 2, 8, 123]	1	[2, 8, 123]
['Пн', 'Вт', 'Четверг']	'Пн'	['Вт', 'Четверг']
[1.25]	1.25	[]
[]	не определена	не определен

В программе голова отделяется от хвоста символом «|».

Часто используемыми операциями при работе со списками являются:

- проверка наличия элемента в списке;
- добавление элемента в список;
- конкатенция (объединение) списков;
- удаление элемента из списка;
- задание обратного порядка следования элементов списка;
- разделение списка на два.

1 Проверка наличия элемента в списке.

Листинг 4.4

```
% 1 параметр – элемент, наличие которого в списке
требуется проверить
% 2 параметр - список
member(X, [X |_]).
member(X, [_|Tail]):- member(X, Tail).


?- member(2, [1, 3, 45, 2, 74]),
write('Да');
write('Нет').
Ответ: Да.
```

2. Добавление элемента в список. Для данной операции не требуется отдельного правила, если элемент X добавляется в начало списка List

```
NewList = [X| List].
```

3. Конкатенция двух списков.

Листинг 4.5

```
% 1 параметр – первый список
% 2 параметр – второй список
% 3 параметр – результат объединения двух списков
concat([], L2, L2).
concat([X|L1], L2, [X|L3]):- concat(L1, L2, L3).

?- concat([1, 2], [3, 45], L),
write(L).
Ответ: [1, 2, 3, 45].
```

4. Удаление элемента из списка и задание обратного порядка следования элементов списка.

Листинг 4.6

```
% 1 параметр – удаляемый элемент
% 2 параметр – исходный список
% 3 параметр – рабочий список
% 4 параметр – перевернутый список без элемента
delete(_, [], L, L).
delete(X, [X|L], L1, L2):- delete(X, L, L1, L2).
delete(X, [Y|L], L1, L2):- X \== Y, delete(X, L, [Y|L1],
L2).

% 1 параметр – исходный список
% 2 параметр – рабочий список
% 3 параметр – перевернутый список
reverse([], Lr, Lr).
reverse([X|L], L1, Lr):- reverse(L, [X|L1], Lr).

?- delete(1, [1, 3, 1, 45, 1], [], L),
reverse(L, Lr),
write(Lr).
Ответ: [3, 45].
```

5. Разделение списка на два.

Листинг 4.7

```
% 1 параметр – элемент, задающий разбиение
% 2 параметр – исходный список
% 3 параметр – элементы, меньшие или равные 1 параметру
% 4 параметр – элементы, большие 1 параметра
split(_, [], [], []).
split(Y, [X|L], [X|L1], L2):- X @<= Y, split(Y, L, L1, L2).
split(Y, [X|L], L1, [X|L2]):- X @> Y, split(Y, L, L1, L2).


?- split(7, [1, 3, 1, 45, 1, 33], L1, L2),
writeln(L1),
writeln(L2).
Ответ: [1, 3, 1, 1] и [45, 33].
```

Постановка задачи к лабораторной работе 4

1. Изучить предлагаемый теоретический материал.
2. Составить программы, используя фрагменты листингов 4.1-4.7.
2. Отчет должен содержать титульный лист, включающий название лабораторной работы и номер варианта, а также сведения об авторе (фамилия студента и номер группы).
3. Оформление каждого задания начинайте с указания его номера, например «Задание № 1» и текста задания.
4. Оформляя решение заданий, следует включать в документ скриншоты и комментарии для пояснения используемых обозначений и выполняемых действий.
5. Оформить отчет по лабораторной работе. Представить отчет по лабораторной работе для защиты.

Варианты индивидуальных заданий

В соответствии с полученным заданием сформировать решение и дать его описание со скриншотами выполненных действий.

Задание 1

Разработать программу расчета функции с использованием рекурсивных правил, отвечающую следующим требованиям (табл.4.1).

1.1. Программа должна запрашивать у пользователя:
- N – количество членов ряда;
- X – значение переменной (если в формуле есть X).

1.2 Результаты работы, выдаваемые на экран:
- приближенное значение функции;
- точное значение функции.

Таблица 4.1 – Программа расчета функции

№	Выражение
1	$y = \frac{1}{1+x} = 1 - x + x^2 - x^3 + \cdots + (-1)^{n-1}x^{n-1}$
2	$y = \frac{1}{(1+x)^2} = 1 - 2x + 3x^2 - 4x^3 + \cdots + (-1)^{n-1}(n+1)x^{n-1}$

3	$y = \frac{1}{1+x^2} = 1 - x^2 + x^4 - x^6 + \cdots + (-1)^{n-1} x^{2n-2}$
4	$y = \sqrt{1 + \sqrt{2 + \cdots + \sqrt{(n-1) + \sqrt{n}}}}$
5	$y = \pi = \sqrt{12}\left(1 - \frac{1}{3 \cdot 3} + \frac{1}{5 \cdot 3^2} - \frac{1}{7 \cdot 3^3} + \cdots + (-1)^{n-1} \frac{1}{(2n-1) \cdot 3^{n-1}}\right)$
6	$y = \frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots + (-1)^{n-1} \frac{1}{2n-1}$
7	$y = \frac{1}{4}(\pi - 3) = \frac{1}{2 \cdot 3 \cdot 4} - \frac{1}{4 \cdot 5 \cdot 6} + \frac{1}{6 \cdot 7 \cdot 8} - \frac{1}{8 \cdot 9 \cdot 10} + \cdots + (-1)^{n-1} \frac{1}{2n(2n+1)(2n+2)}$
8	$y = \frac{\pi^2}{6} = 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \cdots + \frac{1}{n^2}$
9	$y = \frac{\pi^2}{8} = 1 + \frac{1}{3^2} + \frac{1}{5^2} + \frac{1}{7^2} + \cdots + \frac{1}{(2n-1)^2}$
10	$y = \frac{\pi^2}{12} = 1 - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \cdots + (-1)^{n-1} \frac{1}{n^2}$

Задание 2

Разработать программу расчета функции с использованием рекурсивных правил, отвечающую следующим требованиям (табл.4.2).

2.1. Программа должна запрашивать у пользователя:

- N – количество членов ряда;
- X – значение переменной (если в формуле есть X).

2.2 Результаты работы, выдаваемые на экран:

- приближенное значение функции;
- точное значение функции.

Таблица 4.2 – Программа расчета сложной функции

№	Выражение
1	$y = \frac{\pi^3}{32} = 1 - \frac{1}{3^3} + \frac{1}{5^3} - \frac{1}{7^3} + \cdots + (-1)^{n-1} \frac{1}{(2n-1)^3}$
2	$y = \frac{\pi^4}{90} = 1 + \frac{1}{2^4} + \frac{1}{3^4} + \frac{1}{4^4} + \cdots + \frac{1}{n^4}$
3	$y = \frac{\pi^6}{945} = 1 + \frac{1}{2^6} + \frac{1}{3^6} + \frac{1}{4^6} + \cdots + \frac{1}{n^6}$
4	$y = \frac{\pi^8}{9450} = 1 + \frac{1}{2^8} + \frac{1}{3^8} + \frac{1}{4^8} + \cdots + \frac{1}{n^8}$
5	$y = e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^{n-1}}{(n-1)!}$
6	$y = e^{-x} = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots + (-1)^{n-1} \frac{x^{n-1}}{(n-1)!}$
7	$y = e^{\sin x} = 1 + \frac{\sin x}{1!} + \frac{\sin^2 x}{2!} + \frac{\sin^3 x}{3!} + \cdots + \frac{\sin^{n-1} x}{(n-1)!}$
8	$y = e^{-x^2} = 1 - \frac{x^2}{1!} + \frac{x^4}{2!} - \frac{x^6}{3!} + \cdots + (-1)^{n-1} \frac{x^{2n-2}}{(n-1)!}$
9	$y = \ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots + (-1)^{n-1} \frac{x^n}{n}$

10	$y = \ln 2 = 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots + (-1)^{n-1}\frac{1}{n}$

Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

Отчет по лабораторной работе должен состоять из следующих структурных элементов:

титульный лист;
вводная часть;
основная часть (описание работы);
заключение (выводы).

Вводная часть отчета должна включать пункты:

условие задачи;
порядок выполнения.
программно-аппаратные средства, используемые при выполнении работы.

Зашита отчета по лабораторной работе заключается в предъявлении преподавателю полученных результатов в виде файла и демонстрации полученных навыков при ответах на вопросы преподавателя.

Контрольные вопросы

1. Перечислите достоинства и недостатки логического программирования.
2. Перечислите операторы логического программирования.
3. Для чего применяется трассировка?
4. В каких случаях применяются экспертные системы?
5. Чем отличается логическое программирование от функционального?
6. Дайте определение понятия «предикат»
7. Дайте определение регрессии?
8. В каких случаях является целесообразным использование Пролога?
9. Перечислите основные компоненты интеллектуальных систем.
10. В чем отличие базы данных от базы знаний?

Лабораторная работа 5. Разработка экспертных систем

Цель работы: Показать методологию компьютерной реализации математических вычислений, применив инструменты компьютерного моделирования.

Основы теории

Разработка программ на языке Пролог

После запуска интерпретатор Пролога выдает приглашение вводить *запросы*, которые также называют *вопросами* или *целями*. Как правило, запрос представляет собой вызов процедуры и записывается как имя процедуры, за которым в круглых скобках следуют аргументы, разделенные запятыми. В конце запроса ставится точка. Процедуры в Прологе называют *предикатами*. Некоторые из них встроены в язык и их можно вызывать немедленно.

```
?- true.

Yes

?- fail.
```

```
No
```

Встроенные предикаты `true` и `fail` представляют соответственно тождественно истинное и тождественно ложное высказывание. Ответ на первый вопрос всегда "да" , а на второй "нет". Говорят, что запрос `true` заканчивается *успехом*, а `fail` - *неудачей*. Точно так же любой запрос завершается либо успехом, либо неудачей.

Листинг 5.1

```
?- integer(1).


Yes


?- 3+4 < 3*2.


No


?- write('привет').
привет


Yes
```

Обратите внимание: между именем предиката и открывающей скобкой не должно быть пробелов. Второй запрос записан в традиционной инфиксной форме - некоторые предикаты допускают такую запись. Выполнив последний запрос, Пролог отвечает Yes, хотя мы вроде бы ни о чем не спрашивали. Таким образом он просто сообщает о благополучном завершении запроса.

Есть еще одна возможность - запрос может завершиться ошибкой.

Листинг 5.2

```
?- abracadabra.
ERROR: Undefined procedure: abracadabra/0


?- 3/0 > 1.
ERROR: //2: Arithmetic: evaluation error: `zero_divisor'
```

Поскольку системе не известно ничего по поводу `abracadabra`, возникает исключительная ситуация. Во втором случае ошибка возникла в процессе арифметических вычислений. В принципе, ошибка - тоже неудача, но неудача особого рода, после которой дальнейшее выполнение запроса не имеет смысла.

Имея некоторый опыт работы с функциональными языками, мы можем попытаться найти сумму чисел:

```
?- 2 + 3.
ERROR: Undefined procedure: (+)/2
```

и получаем сообщение об ошибке. Наш запрос не имеет формы предложения. Попробуем по другому.

```
?- write(2+3).


2+3
Yes
```

Система выводит выражение, даже не пытаясь что-нибудь вычислить. Может быть так?

```
?- X = 2+3.


X = 2+3
Yes
```

Снова неудача. Пролог разрабатывался прежде всего как язык обработки символьной информации и выражения вида "2+3" - это его объекты данных (они называются *термами*). Именно это выражение и присваивается переменной X. Но надо же как то выполнять арифметические вычисления. Для этого предназначены специальные *арифметические предикаты*.

Листинг 5.3

```
?- X is 2+3.


X = 5
Yes
```

Наконец-то! Предикат `is` вычисляет значение выражения и присваивает его переменной. Запрос завершается успехом и значение переменной выводится в качестве ответа.

Другие арифметические предикаты (`=:=`, `=\=`, `<`, `=<`, `>`, `>=`) вычисляют оба своих аргумента и сравнивают их значения.

Листинг 5.4

```
?-  3 + 2 =:= 5.


Yes


?- 2 * 2 =:= 2 + 2 .


Yes


?- exp(2) >= sin(2) .


Yes
```

Другая разновидность примитивных объектов - *атомы*. Чаще всего они записываются как последовательности букв и цифр, начинающейся со *строчной* буквы. Это отличает их от переменных, которые начинаются с *прописной* буквы. Атомы можно сравнивать посредством предикатов `==` , `\==`, `@<`, `@=<`, `@>`, `@>=` . Арифметические предикаты для этого не подходят, поскольку они попытаются вычислить арифметические значения атомов, что приведет к ошибке. А вот числа (но не значения выражений) можно сравнивать и теми и другими.

Листинг 5.5

```
?- a @< b.
Yes

?- 1 @< 2.
Yes

?- 2 < a.
ERROR: Arithmetic: `a/0' is not a function

?- 2 < e.
Yes
```

Почему последний запрос успешен, догадайтесь сами.

```
?- X is 2+3.


X = 5
Yes

?- Y is X+1.
ERROR: Arguments are not sufficiently instantiated
```

В сообщение об ошибке указано что переменная(имеется в виду X) не конкретизирована. Но мы же только что присвоили ей значение! Дело в том, что это уже другая переменная. Область действия переменной распространяется только на один запрос. Но запросы могут быть и сложными, состоящими из нескольких целей.

Листинг 5.6

```
?- X is 2+3, Y is X+1.

X = 5
Y = 6

Yes
```

Пока наша возможность задавать вопросы ограничена. Чтобы задавать более интересные вопросы, надо загрузить какую-нибудь программу. Это можно процедурой `consult`, для которой придумано сокращение. Например, если программа находиться в файле 'my_program.pl', то ее можно загрузить, введя

```
?- consult(my_program).
```

или

```
?- [my_program].
```

А команда

```
?- [my_program,another_program,yet_another_program].
```

загрузит сразу все указанные программы. Но прежде чем загружать программу надо ее написать.

Программирование на языке Пролог

Программы на Прологе часто называют "базами данных" или даже "базами знаний" в том смысле, что они представляет собой совокупности предложений, определяющих отношения между объектами предметной области или свойства этих объектов. Свойства и отношения в Прологе, как и в логике, называют предикатами.

Каждый предикат определяется последовательностью *предложений*, которую называют *процедурой*. Традиционно выделяют два типа предложений: *факты* и *правила*. Для начала разберемся с первой разновидностью. Предложение-факт состоит из имени предиката, за которым в круглых скобках следуют аргументы. Такое предложение описывает утверждение о конкретных объектах. Программа, состоящая только из фактов в сущности является реляционной базой данных.

Возьмем в качестве примера фрагмент базы данных небольшой компании Horns&Hoofs Ltd. Одноместный предикат `employee` устанавливает свойство "быть работником".

Листинг 5.7.1

```
employee(anthony).
employee(barbara).
employee(charles).
employee(diana).
employee(edward).
employee(frederic).
employee(gregory).
employee(herbert).
employee(isabella).
employee(john).
```

Вообще то, имена собственные положено писать с прописной буквы, но в Прологе имена отношений и объектов должны начинаться со строчной буквы или заключаться в кавычки. Как правило, предпочитают обходится без кавычек. Программа может содержать предикаты с одинаковыми именами, но с разным количеством аргументов. Чтобы различать их употребляют имена состоящие из имени предиката и его арности. Полное имя только что определенного предиката: `employee/1`.

Предикат `salary/2` устанавливает оклад каждого работника.

Листинг 5.7.2

```
salary(anthony,500).
salary(barbara,200).
salary(charles,180).
salary(diana, 150).
salary(edward,150).
salary(frederic,140).
salary(gregory,150).
salary(herbert,100).
salary(isabella,90).
salary(john,160).
```

Предикат `boss/2` устанавливает описывает организационную структуру компании. Утверждение boss(A,B) означает, что B - руководитель A.

Листинг 5.7.3

```
boss(barbara, anthony).
boss(charles, anthony).
boss(diana,   barbara).
boss(edward,  barbara).
boss(frederic,charles).
boss(gregory, charles).
boss(herbert, charles).
boss(isabella,gregory).
boss(john,    gregory).
```

Загрузив программу, можно задавать вопросы, непосредственно к ней относящиеся.

Листинг 5.7.4

```
?- [hh].
% hh compiled 0.00 sec, 0 bytes
Yes

?- employee(anthony).
Yes

?- employee(ronald).
No

?- boss(barbara, anthony).
```

```
Yes
```

Это простейший тип вопросов. Система просто проверяет записано ли некоторое предложение в программе и отвечает "да" если это так или "нет" в противном случае. Таким образом, предложение считается ложным, если явно не записано обратного.

Более интересные вопросы получаются, если подставить в запросы переменные. Например, мы можем узнать зарплату Чарльза, спросив.

```
?- salary(charles,S).


S = 180
Yes
```

Отвечая на этот вопрос Пролог сопоставляет его со всеми фактами из базы данных, пытаясь подобрать подходящее значение переменной. Некоторые вопросы допускают несколько вариантов ответа. Такие вопросы, называют *недетерминированными*.

```
?- boss(A,charles).


A = frederic
Yes
```

Но это не единственный ответ. Отвечая на вопросы с переменными система, выдав значение ждет нажатия на Enter. На самом деле, она интересуется нужны ли нам другие ответы. Нажимая на Enter мы говорим "спасибо, достаточно" и система отвечает *Yes*. Чтобы получить другие ответы надо ввести ";".

```
?- boss(A,charles).


A = frederic ;
A = gregory ;
A = herbert ;
No
```

Последнее *No* означает, что других ответов нет. Ответы выдаются в том же порядке, в котором они встречаются в программе. Вопрос с двумя переменными вернет все пары подходящие значений.

```
?- boss(A,B).


A = barbara
B = anthony ;


A = charles
B = anthony ;


A = diana
B = barbara
```

```
Yes
```

В тех случаях, когда требуется указать переменную но её значение для нас несущественно можно использовать так называемую *анонимную переменную*, состоящую из единственного символа подчеркивания. Отвечая на вопрос, содержащий анонимную переменную, Пролог подбирает для нее значение, но не выводит его.

```
?- boss(A,_).

A = barbara ;
A = charles ;
A = diana ;
A = edward
Yes
```

До сих пор мы задавали довольно простые вопросы. Чтобы ответить на них достаточно было просмотреть одно отношение. Но как мы знаем, простые вопросы можно объединять в более сложные. Запятая при этом получает логический смысл конъюнкции предложений. Например, спросим не получает ли кто то больше своего начальника

```
?- boss(A,B), salary(A,SA), salary(B,SB), SA > SB.

A = john
B = gregory
SA = 160
SB = 150 ;

No
```

и обнаружим непорядок: оклад Джона больше, чем у Грегори.

Кроме конъюнкции для соединения предложений можно применять и дизъюнкцию, которая обозначается точкой с запятой ";".

```
?- salary(X,S),(S<100;S>200).

X = anthony
S = 500 ;

X = isabella
S = 90 ;

No
```

Отрицание обозначается "\+" (или `not`, но эта форма, хотя и привычная, не рекомендуется).

```
?-  \+ employee(X).
```

```
No
```

Запрос означает "не существует ни одного работника", что, конечно, ложно.

```
?- \+ (employee(X),salary(X,300)).

X = _G157
Yes
```

А этот запрос "не существует ни одного работника с окладом 300". Таких работников, действительно, нет. Переменная X осталась свободной (не получила никакого значения), поэтому в качестве значения выводится нечто маловразумительное.

Как видно, отрицание позволяет формулировать общие (общеотрицательные в терминологии классической логики) суждения. Выясним, кто не является начальником и кто не имеет начальников.

```
?- employee(X), \+ boss(_,X).

X = diana ;
X = frederic ;
X = herbert ;
X = isabella ;
X = john ;
No

?- employee(X), \+ boss(X,_).


X = anthony ;
No
```

Найдем самого высокооплачиваемого работника.

```
?- salary(A,Smax), \+ (salary(_,S), S>Smax).

A = anthony
Smax = 500
S = _G161

Yes
```

Интересное применение отрицания - конструкция (`\+ (Q, \+ P)`), которая проверяет, что все решения Q будут решениями для P. Например

```
?- \+ (employee(X), \+ salary(X,_)).
```

```
X = _G157
Yes
```

Это буквально означает "не существует работника, не имеющего оклада" или "все работники имеют оклады". В SWI-Prolog есть встроенный предикат `forall`, имеющий тот же смысл.

```
?- forall(employee(X), salary(X,_)).

X = _G157
Yes
```

Кроме фактов - безусловно истинных предложений, в программы можно включать и условные предложения - *правила*, которые определяют новые предикаты в терминах уже существующих. Каждое правило состоит из двух частей, разделенных символом ":-". Левая часть правила - *заголовок* описывает предикат, а правая - *тело* определяет условия, при которых он истинен. Телом может быть любой запрос, который и выполняется при вызове предиката, определенного в заголовке. Например, можно оформить в виде правил, вопросы, которые мы задавали.

```
all_paid :- forall(employee(X), salary(X,_)).

costly(A) :- boss(A,B), salary(A,SA), salary(B,SB), SA >
SB.

noboss(A) :- employee(A), \+ boss(_,A).

boss(B) :-  boss(_,B).
```

Дополнив программу новыми правилами, сможем задавать новые вопросы. При этом не интересующие нас переменные оказываются спрятанными внутри правил и не выводятся.

```
?- all_paid.

Yes


?- costly(A).

A = john ;
No


?- noboss(A).
```

```
A = diana ;
A = frederic ;
A = herbert ;
A = isabella ;
A = john ;
No

?- boss(charles).

Yes

?- boss(B).

B = anthony ;
B = anthony ;
B = barbara ;
B = barbara ;
B = charles ;
B = charles ;
B = charles ;
B = gregory ;
B = gregory ;
No
```

Обратите внимание, что в последнем запросе каждый ответ выдается несколько раз, ровно столько, сколько он встречается в программе. Нам еще придется столкнуться с этой особенностью - Пролог выдает ответ столько раз, сколькими способами он может его найти.

Постановка задачи к лабораторной работе 5

1. Изучить предлагаемый теоретический материал.
2. Составить программы, используя фрагменты листингов 5.1-5.6.
2. Отчет должен содержать титульный лист, включающий название лабораторной работы и номер варианта, а также сведения об авторе (фамилия студента и номер группы).
3. Оформление каждого задания начинайте с указания его номера, например «Задание № 1» и текста задания.
4. Оформляя решение заданий, следует включать в документ скриншоты и комментарии для пояснения используемых обозначений и выполняемых действий.
5. Оформить отчет по лабораторной работе. Представить отчет по лабораторной работе для защиты.

Варианты индивидуальных заданий

В соответствии с полученным заданием сформировать решение и дать его описание со скриншотами выполненных действий.

Ввести факты и правила из листингов 5.7.1-5.7.4. Сформировать запросы (табл.5.1).

1.1. Программа должна запрашивать у пользователя:

Запрос, сформулированный в таблице.

1.2 Результаты работы, выдаваемые на экран:

Ответ на запрос.

Таблица 5.1 – Запросы

№	Выражение
1	Найти зарплату Чарльза
2	Найти зарплаты сотрудников
3	Выяснить, чья зарплата больше
4	Найти самого высокооплачиваемого работника
5	Кто зарабатывает больше начальника?
6	Кто не является начальником и кто не имеет начальников
7	Найти зарплаты Дианы и Фредерика
8	Сравнить зарплаты Герберта и Джона
9	Найти зарплаты Дианы, Фредерика, Герберта и Джона
10	Сравнить зарплаты Герберта, Джона, Дианы и Фредерика

Содержание отчета

По выполненной работе составляется отчет. Отчет выполняется в электронном виде. По выполненному отчету проводится защита лабораторной работы.

Отчет по лабораторной работе должен состоять из следующих структурных элементов:

титульный лист;

вводная часть;

основная часть (описание работы);

заключение (выводы).

Вводная часть отчета должна включать пункты:

условие задачи;

порядок выполнения.

программно-аппаратные средства, используемые при выполнении работы.

Зашита отчета по лабораторной работе заключается в предъявлении преподавателю полученных результатов в виде файла и демонстрации полученных навыков при ответах на вопросы преподавателя.

Контрольные вопросы

1. Как представляется список в Прологе?
2. Приведите примеры списков на Прологе
3. Что такое рекурсия?
4. Для чего используется представление списка в виде головы и хвоста?
5. Какое минимальное количество предложений Пролога необходимо для программирования рекурсии?
6. В чем преимущество использования рекурсии?
7. Функциональность экспертных систем
8. Функциональность СППР

5. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

5.1. Перечень основной и дополнительной литературы, необходимой для освоения дисциплины

5.1.1. Перечень основной литературы

1. Цуканова Н.М, Дмитриева Т.А. Логическое программирование на языке Visual Prolog. Серия: Специальность. Для высших учебных заведений – М.: Горячая линия-Телеком, 2016.

2. Макконнелл, С. Совершенный код. Мастер-класс. / Стив Макконнелл. - М.: Русская Редакция, 2016.

5.1.2. Перечень дополнительной литературы

1. Джозеф Джарратано, Гари Райли. Экспертные системы. Принципы разработки и программирование, 4-е издание. – М.: Вильямс, 2017.

2. Чезарини Ф., Томпсон С. Программирование в Erlang. М.: ДМК Пресс, 2016.

5.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине

1. Методические указания по выполнению лабораторных работ по дисциплине «Введение в функциональное программирование».

2. Методические рекомендации для студентов по организации самостоятельной работы по дисциплине «Введение в функциональное программирование».

5.3. Перечень ресурсов информационно-телекоммуникационной сети Интернет, необходимых для освоения дисциплины

1. Национальный Открытый Университет. Интуит. http://www.intuit.ru.
2. Федеральный портал «Российское образование. http://www.edu.ru.
3. Российская государственная библиотека. http://www.rsl.ru.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

Методические указания
по выполнению самостоятельных работ по дисциплине
«ВВЕДЕНИЕ В ФУНКЦИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ»

направления подготовки **09.03.02 Информационные системы и технологии**

Направленность (профиль): **Информационные системы и технологии обработки цифрового контента**

Пятигорск, 2025г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Методические рекомендации содержат перечень тем с вопросами для самостоятельной проработки, перечень лабораторных работ с вопросами для самостоятельной проработки.

Методические указания посвящены курсу «Введение в функциональное программирование». На протяжении многих тысячелетий человечество занимается накоплением, обработкой и передачей знаний. Для этих целей непрерывно изобретаются новые средства и совершенствуются старые: речь, письменность, почта, телеграф, телефон и т. д. Большую роль в технологии обработки знаний сыграло появление компьютеров.

В октябре 1981 года Японское министерство международной торговли и промышленности объявило о создании исследовательской организации — Института по разработке методов создания компьютеров нового поколения (Institute for New Generation Computer Technology Research Center). Целью данного проекта было создание систем обработки информации, базирующихся на знаниях. Предполагалось, что эти системы будут обеспечивать простоту управления за счет возможности общения с пользователями при помощи естественного языка. Эти системы должны были самообучаться, использовать накапливаемые в памяти знания для решения различного рода задач, предоставлять пользователям экспертные консультации, причем от пользователя не требовалось быть специалистом в информатике. Предполагалось, что человек сможет использовать ЭВМ пятого поколения так же легко, как любые бытовые электроприборы типа телевизора, магнитофона и пылесоса. Вскоре вслед за японским стартовали американский и европейский проекты.

Появление таких систем могло бы изменить технологии за счет использования баз знаний и экспертных систем. Основная суть качественного перехода к пятому поколению ЭВМ заключалась в переходе от обработки данных к обработке знаний. Японцы надеялись, что им удастся не подстраивать мышление человека под принципы функционирования компьютеров, а приблизить работу компьютера к тому, как мыслит человек, отойдя при этом от фон неймановской архитектуры компьютеров. В 1991 году предполагалось создать первый прототип компьютеров пятого поколения.

Теперь уже понятно, что поставленные цели в полной мере так и не были достигнуты, однако этот проект послужил импульсом к развитию нового витка исследований в области искусственного интеллекта и вызвал взрыв интереса к логическому программированию. Так как для эффективной реализации традиционная фон неймановская архитектура не подходила, были созданы специализированные компьютеры логического программирования PSI и PIM.

В качестве основной методологии разработки программных средств для проекта ЭВМ пятого поколения было избрано логическое программирование, ярким представителем которого является язык Пролог. Думается, что и в настоящее время Пролог остается наиболее популярным языком искусственного интеллекта в Японии и Европе (в США, традиционно, более распространен другой язык искусственного интеллекта —язык функционального программирования Лисп).

Название языка "Пролог" происходит от слов ЛОГическое ПРОграммирование (PROgrammation en LOGique во французском варианте и PROgramming in LOGic — в английском).

Пролог основывается на таком разделе математической логики, как исчисление предикатов. Точнее, его базис составляет процедура доказательства теорем методом резолюции для хорновских дизъюнктов.

1. ЦЕЛЬ И ЗАДАЧИ ИЗУЧЕНИЯ ДИСЦИПЛИНЫ

Цель освоения дисциплины «Введение в функциональное программирование» формирование набора профессиональных компетенций будущего бакалавра по направлению подготовки 09.03.02 «Информационные системы и технологии».

Задачи освоения дисциплины: изучение основных понятий функционального программирования, освоение инструментов решения задач функционального программирования.

2. РЕКОМЕНДАЦИИ ДЛЯ САМОПОДГОТОВКИ

2.1 Подготовка к лекциям. Самостоятельное изучение литературы.

Базовый уровень

Тема 1. История развития логического программирования

1. Факты: определение, назначение, примеры
2. Основные и неосновные факты
3. Вопросы: определение, назначение, примеры

Тема 2. Основы математической логики

4. Основные и неосновные вопросы
5. Правила: определение, назначение, примеры
6. Определение логической программы и логического следствия

Тема 3. Исчисление высказываний и предикатов

7. Термы, виды термов
8. Понятия корректности и сложности логической программы
9. Экзистенциальные вопросы, правило обобщения

Тема 4. Применение логического программирования в проектировании экспертных систем

10. Универсальные факты, правило конкретизации
11. Конъюнктивные вопросы
12. Унификатор двух термов, унифицируемые термы

Тема 5. Введение в программирование на языке PROLOG

13. Наибольший общий унификатор.
14. Алгоритм унификации.
15. Абстрактный интерпретатор. Определение вычисления цели

Тема 6. Структура программы на языке PROLOG. Синтаксис языка PROLOG

16. Предикаты в языке Пролог
17. Предложения в языке Пролог: факты и правила
18. Запросы (цели) в языке Пролог

Тема 7. Решение задач на вычисление цели

19. Синтаксис языка Пролог. Переменные.
20. Сопоставление и унификация. Предикат равенства. Детерминизм
21. Основные принципы поиска с возвратом

Тема 8. Поиск решения в задачах логического программирования

22. Управление поиском решений (предикаты fail и !)
23. Управление поиском решений (динамическое отсечение)
24. Простые объекты данных. Составные объекты данных. Составные объекты данных
25. Предикат repeat
26. Рекурсия. Хвостовая рекурсия
27. Динамические базы данных
28. Списки, примеры работы. Сортировка списков
29. Деревья, примеры работы

30. Графы: представление и действия над графами

Повышенный уровень

Тема 1. История развития логического программирования

1. Абстрактный интерпретатор. Производные и порожденные цели. Цели-предшественники и родственные цели
2. Алгоритм работы абстрактного интерпретатора
3. Декларативный смысл пролог-программ
4. Процедурная семантика пролог-программ

Тема 2. Основы математической логики

5. Представление списков, голова и хвост списка
6. Основные предикаты для работы со списками
7. Предикат findall
8. Задача о недетерминированном конечном автомате

Тема 3. Исчисление высказываний и предикатов

9. Ограничение перебора, отсечение
10. Цели fail и true
11. Отрицание в Прологе
12. Обработка строк, основные предикаты

Тема 4. Применение логического программирования в проектировании экспертных систем

13. Структура программы в Visual Prolog
14. Объявление пользовательских доменов, альтернативные домены
15. Описание предикатов в Visual Prolog

Тема 5. Введение в программирование на языке PROLOG

16. Управление детерминизмом в Visual Prolog
17. Описание динамических баз фактов
18. Предикаты для работы с динамическими базами фактов
19. Управление детерминизмом в базах фактов

Тема 6. Структура программы на языке PROLOG. Синтаксис языка PROLOG

20. Задача о восьми ферзях в трех вариантах
21. Основные предикаты для обработки строк
22. Общие подходы к решению головоломок
23. Задача о ханойской башне

Тема 7. Решение задач на вычисление цели

24. Задача о крестьянине, волке, козе и капусте
25. Определение пространства состояний
26. Примеры определения отношения после для разных задач
27. Поиск в глубину в пространстве состояний

Тема 8. Поиск решения в задачах логического программирования

28. Поиск в глубину с обнаружением циклов
29. Поиск в глубину с ограничением глубины
30. Поиск в ширину в пространстве состояний. Представление путей-кандидатов в виде списков

2.2 Подготовка к лабораторным работам

Базовый уровень

Тема 1: Введение в программирование на языке PROLOG

1. Автоматизация доказательства в логике предикатов
2. Метод резолюций в исчислении высказываний
3. Сравнительная характеристика современных телекоммуникационных систем.
4. Правило унификации в логике предикатов
5. Метод резолюций в исчислении предикатов

Тема 2: Структура программы на языке PROLOG. Синтаксис языка PROLOG

6. Процедурность Пролога
7. Структура программ Пролога
8. Основы синтаксиса языка Пролог
9. Использование дизъюнкции и отрицания
10. Унификация в Прологе

Тема 3. Алгоритмы логического программирования. Решение задач на рекурсию

11. Изучить синтаксис и семантику операций математической логики
12. Дать описание задач математической логики с использованием термов и формул
13. Изучить правила написания программ на языке Пролог
14. Провести простые вычисления на языке Пролог
15. Выполнить операции над списками с использованием языка Пролог

Тема 4. Применение списков в Прологе

16. Составить алгоритмы для задач логического программирования
17. Выполнить поиск элемента в списке с использованием языка Пролог
18. Выполнить объединение двух списков использованием языка Пролог
19. Выполнить определение размерности списка с использованием языка Пролог
20. Выполнить определение максимального элемента в списке
21. Задача о крестьянине, волке, козе и капусте
22. Определение пространства состояний
23. Примеры определения отношения после для разных задач
24. Поиск в глубину в пространстве состояний

Повышенный уровень

Тема 1: Введение в программирование на языке PROLOG

1. Основные стратегии поиска решений в пространстве состояний
2. Вычисление цели в задачах логического программирования
3. Управление поиском решения в задачах логического программирования
4. Применение списков в задачах логического программирования

Тема 2: Структура программы на языке PROLOG. Синтаксис языка PROLOG

5. Исчисление высказываний и предикатов
6. Операции логического программирования
7. Поиск в глубину
8. Поиск в ширину
9. Решение игровых задач
10. Минимаксный принцип поиска решений

Тема 3. Алгоритмы логического программирования. Решение задач на рекурсию

11. Разработать стратегию поиска решений в пространстве состояний.
12. Выполнить поиск в глубину
13. Выполнить поиск в ширину

Тема 4. Применение списков в Прологе

14. Разработать решение игровых задач
15. Прменить минимаксный принцип поиска решений
16. Разработать экспертную систему для проверки уровня знаний
17. Разработать экспертную систему для оценки работоспособности аппаратно-программной инфраструктуры

18. Разработать экспертную систему для оценки качества программирования
19. Разработать интеллектуальный сервис для диагностики неисправности аппаратно-программной инфраструктуры
20. Разработать интеллектуальный сервис для подсказки в определении профиля санатория

3. ТЕОРЕТИЧЕСКИЙ МАТЕРИАЛ

3.1 ВВЕДЕНИЕ В ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ PROLOG

Императивные и декларативные языки программирования

Традиционно под программой понимают последовательность операторов (команд, выполняемых компьютером). Этот стиль программирования принято называть императивным. Программируя в императивном стиле, программист должен объяснить компьютеру, как нужно решать задачу.

Противоположный ему стиль программирования — так называемый декларативный стиль, в котором программа представляет собой совокупность утверждений, описывающих фрагмент предметной области или сложившуюся ситуацию. Программируя в декларативном стиле, программист должен описать, что нужно решать.

Соответственно и языки программирования делят на императивные и декларативные.

Императивные языки основаны на фон неймановской модели вычислений компьютера. Решая задачу, императивный программист вначале создает модель в некоторой формальной системе, а затем переписывает решение на императивный язык программирования в терминах компьютера. Но, во-первых, для человека рассуждать в терминах компьютера довольно неестественно. Во-вторых, последний этап этой деятельности (переписывание решения на язык программирования) по сути дела не имеет отношения к решению исходной задачи. Очень часто императивные программисты даже разделяют работу в соответствии с двумя описанными выше этапами. Одни люди, постановщики задач, придумывают решение задачи, а другие, кодировщики, переводят это решение на язык программирования.

В основе декларативных языков лежит формализованная человеческая логика. Человек лишь описывает решаемую задачу, а поиском решения занимается императивная система программирования. В итоге получаем значительно большую скорость разработки приложений, значительно меньший размер исходного кода, легкость записи знаний на декларативных языках, более понятные, по сравнению с императивными языками, программы.

Известна классификация языков программирования по их близости либо к машинному языку, либо к естественному человеческому языку. Те, что ближе к компьютеру, относят к языкам низкого уровня, а те, что ближе к человеку, называют языками высокого уровня. В этом смысле декларативные языки можно назвать языками сверхвысокого или наивысшего уровня, поскольку они очень близки к человеческому языку и человеческому мышлению.

К императивным языкам относятся такие языки программирования, как Паскаль, Бейсик, Си и т. д. В отличие от них, Пролог является декларативным языком.

Программирование на языке Пролог

При программировании на Прологе усилия программиста должны быть направлены на описание логической модели фрагмента предметной области решаемой задачи в терминах объектов предметной области, их свойств и отношений между собой, а не деталей программной реализации. Фактически Пролог представляет собой не столько язык для программирования, сколько язык для описания данных и логики их обработки. Программа на Прологе не является таковой в классическом понимании, поскольку не содержит явных управляющих конструкций типа условных операторов, операторов цикла

и т. д. Она представляет собой модель фрагмента предметной области, о котором идет речь в задаче. И решение задачи записывается не в терминах компьютера, а в терминах предметной области решаемой задачи, в духе модного сейчас объектно-ориентированного программирования.

Пролог очень хорошо подходит для описания взаимоотношений между объектами. Поэтому Пролог называют реляционным языком. Причем "реляционность" Пролога значительно более мощная и развитая, чем "реляционность" языков, используемых для обработки баз данных. Часто Пролог используется для создания систем управления базами данных, где применяются очень сложные запросы, которые довольно легко записать на Прологе.

В Прологе очень компактно, по сравнению с императивными языками, описываются многие алгоритмы. По статистике, строка исходного текста программы на языке Пролог соответствует четырнадцати строкам исходного текста программы на императивном языке, решающем ту же задачу. Пролог-программу, как правило, очень легко писать, понимать и отлаживать. Это приводит к тому, что время разработки приложения на языке Пролог во многих случаях на порядок быстрее, чем на императивных языках. В Прологе легко описывать и обрабатывать сложные структуры данных. Проверим эти утверждения на собственном опыте при изучении данного курса.

Прологу присущ ряд механизмов, которыми не обладают традиционные языки программирования: сопоставление с образцом, вывод с поиском и возвратом. Еще одно существенное отличие заключается в том, что для хранения данных в Прологе используются списки, а не массивы. В языке отсутствуют операторы присваивания и безусловного перехода, указатели. Естественным и зачастую единственным методом программирования является рекурсия.

Для решения любой задачи есть оптимальный язык (языки) программирования. Многие задачи, хорошо решаемые императивными языками типа Паскаля и Си, плохо решаются на Прологе, и наоборот. Основные области применения Пролога:

быстрая разработка прототипов прикладных программ;

автоматический перевод с одного языка на другой;

создание естественно-языковых интерфейсов для существующих систем;

символьные вычисления для решения уравнений, дифференцирования и интегрирования;

проектирование динамических реляционных баз данных;

экспертные системы и оболочки экспертных систем;

автоматизированное управление производственными процессами;

автоматическое доказательство теорем;

полуавтоматическое составление расписаний;

системы автоматизированного проектирования, базирующееся на знаниях программное обеспечение;

организация сервера данных или, точнее, сервера знаний, к которому может обращаться клиентское приложение, написанное на каком-либо языке программирования.

Знакомство с интерпретатором SWI-PROLOG

В папке с установленным SWI-PROLOG войдите в директорию pl/bin, содержащую файл plwin.exe, и запустите его. На экране появится главное меню и главное (диалоговое) окно с приглашением SWI-PROLOG (см рис.1.1).

Главное меню можно сделать активным, нажав F10 или Alt. Когда главное меню активно, его элементы можно выбрать с помощью клавиш управления курсором и последующим нажатием клавиши Enter. Выбирать элементы главного меню можно также и мышью.

Программа на Прологе

Программа на Прологе состоит из фактов и правил, которые образуют базу знаний Пролог-программы, и запроса к этой базе, который задает цель поиска решений.

Предикаты

Описывают отношение между объектами, которые являются аргументами предиката.

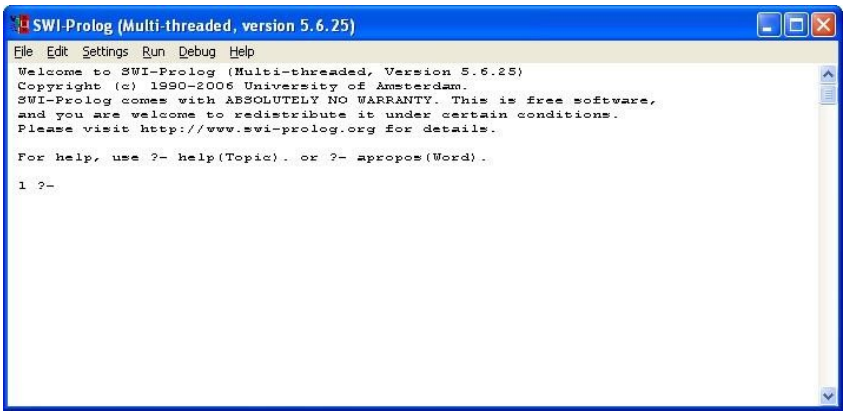

Рисунок 1.1 Вид диалогового окна SWI-PROLOG

Факты

Констатируют наличие заданного предикатом отношения между указанными объектами.

ПРИМЕР

Констатация факта в предложении
Эллен любит теннис.

в синтаксисе Пролога выглядит так:

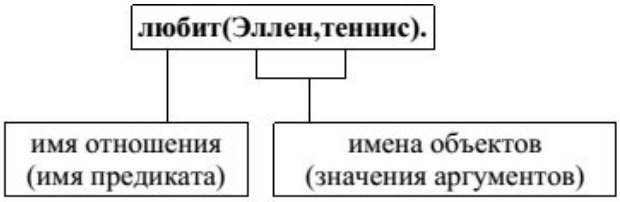

Рисунок 1.2 – Синтаксис Пролог

Имя предиката (функтора) и объекта должно начинаться с маленькой буквы и может содержать латинские буквы, кириллицу, цифры и символ подчеркивания (_). Кириллица используется наравне с латинскими буквами. Обычно предикатам дают такие имена, чтобы они отражали смысл отношения. Например: main, add_file_name. Два предиката могут иметь одинаковые имена, тогда система распознает их как разные предикаты, если они имеют различное число аргументов (арность). Например, любит/2, любит/3.

Имя предиката может совпадать с именем какого-либо встроенного предиката SWI-PROLOG. Однако, если совпали имена пользовательского и встроенного предиката, то при обращении к нему (либо из интерпретатора, либо из программы), будет вызван пользовательский предикат, т.е. пользовательское определение «перекроет» предопределенное в SWI-PROLOG.

Правила

Правила описывают связи между предикатами.

```
Билл любит все, что любит Том.
```

в синтаксисе Пролога:

```
любит('Билл', Нечто):- любит('Том',Нечто).
```

Правило B:-A соответствует импликации A→B («ЕСЛИ A , ТО B»).

В общем виде правило - это конструкция вида:

P0:-P1,P2,…,Pn.

которая читается «P0 истинно, если P1 и P2 и ... Pn истинны».

Предикат P0 называется заголовком правила, выражение P1,P2,…,Pn - телом правила, а предикаты Pi - подцелями правила. Запятая означает логическое "И".

Факты и правила называются также утверждениями или клозами. Факт можно рассматривать как правило, имеющее заголовок и пустое тело.

Процедура

Процедура это совокупность утверждений, заголовки которых имеют одинаковый функтор и одну и ту же арность. Процедура задает определение предиката.

Конец предложения всегда отмечается точкой, поэтому все факты, правила и запросы должны заканчиваться точкой. Заметим также, что между именем предиката и скобкой не должно быть пробелов.

Переменная

Переменная - поименованная область памяти, где может храниться значение.

Если переменная не связана со значением – она называется свободной переменной.

Унификация

Унификация - процесс получения свободной переменной значения в результате сопоставления при логическом выводе в SWI-PROLOG.

Понятие переменной в логическом программировании отличается от базового понятия переменной, которое вводится в структурном программировании. Прежде всего, это отличие заключается в том, что переменная в SWI-PROLOG, однажды получив свое значение при унификации в процессе работы программы, не может его изменить, т.е. она скорее является аналогом математического понятия «переменная» – неизвестная величина. Переменная в SWI-PROLOG не имеет предопределенного типа данных и может быть связана со значением любого типа данных.

Переменная в SWI/PROLOG обозначается как последовательность латинских букв, кириллицы и цифр, начинающаяся с заглавной буквы или символа подчеркивания (_). Заметим, что если значение аргумента предиката или его имя начинается с заглавной буквы, то оно пишется в апострофах.

В SWI-PROLOG различаются строчные и заглавные буквы.

Рассмотрим следующую программу на SWI-PROLOG, которую будем использовать для иллюстрации процессов создания, выполнения и редактирования Пролог-программ.

Листинг 1.1.

```
/* кто что любит */
любит('Эллен', теннис). %Эллен любит теннис
любит('Джон', футбол). %Джон любит футбол
любит('Том', бейсбол). %Том любит бейсбол
любит('Эрик', плавание). %Эрик любит плавание
любит('Марк', теннис). %Марк любит теннис
любит('Билл',X):-любит('Том',X).
```

```
%Билл любит то, что любит Том
```

Комментарий в строке программы начинается с символа % и заканчивается концом строки. Блок комментариев выделяется специальными скобками:

/* (начало) и */ (конец).

Задание 1.1

Для того чтобы набрать текст программы, воспользуйтесь встроенным текстовым редактором. Чтобы создать новый файл выберете команду File/New, в диалоговом окне укажите имя нового файла, например, тест. Для редактирования уже созданного файла с использованием встроенного редактора можно воспользоваться командой меню File/Edit. Набейте программу 1 (текст программы выше по тексту) и сохраните ее (File/Save buffer).

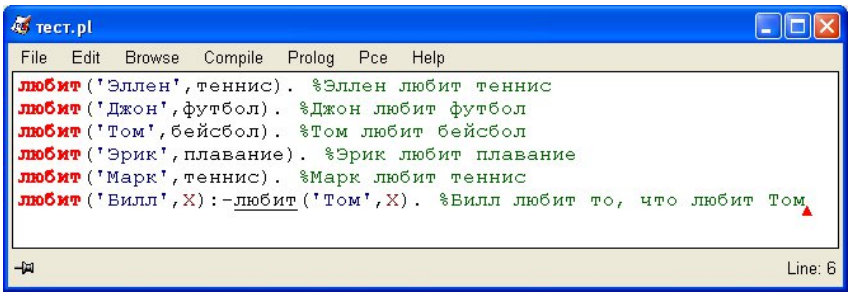

Рисунок 1.2 Внешний вид редактора

Красным цветом подсвечиваются предикаты в заголовках предложений, которые с точки зрения синтаксиса SWI-PROLOG корректны. Указатель “курсор” можно использовать для выверки (например, корректности) расстановки скобок. Зелёным цветом выделяются комментарии, темно-красным цветом - переменные. Подчеркиванием выделяются предикаты в теле правила, которые совпадают с предикатом заголовка,- таким образом акцентируется внимание на возможном зацикливании программы.

Чтобы запустить программу, сначала необходимо ее загрузить в SWI-PROLOG для выполнения. Это делается выбором опции Compile/Compile buffer из окна редактора. Результат компиляции отображается в окне интерпретатора SWI-PROLOG. Там же указываются ошибки, возникшие при компиляции, чаще всего они отображаются и во всплывающем окне ошибок. Обычно перед компиляцией предлагается сохранить файл.

Другой способ загрузить уже существующий файл – это выполнение команды Consult в подменю File диалогового окна SWI-PROLOG. На экране появится диалоговое окно.

Рисунок 1.3 Диалоговое окно

Укажите имя файла, который вы хотите загрузить, и выберите Открыть. Если вы попытаетесь загрузить для выполнения файл, в котором есть синтаксические ошибки, то он не загрузится, а вы получите сообщение об ошибке в главном окне. Угловые скобки << >> будут выделять место, где встретилась ошибка. По умолчанию файлы, ассоциируемые с SWI-PROLOG имеют расширение .pl.

Файлы также можно загрузить, используя встроенный предикат:

```
consult(Имя файла или имена нескольких файлов).
```

Листинг 1.2

```
consult(Test).   % test – имя файла
consult([Test1,Test2]).    % Загрузка двух файлов.
consult('test.pl').1
```

Для выполнения загрузки этот предикат нужно написать в главном окне после приглашения интерпретатора (?-), которое означает, что интерпретатор ждет запрос.

Запрос

Запрос это конструкция вида:

?- P1,P2,…,Pn.

которая читается "Верно ли P1 и P2 и ... Pn ?". Предикаты Pi называются подцелями запроса.

Запрос является способом запуска механизма логического вывода, т.е фактически запускает Пролог-программу.

Для просмотра предложений загруженной базы знаний можно использовать встроенный предикат listing.

Проверьте загрузку исходного файла (Листинг 1.1), задайте запрос:

```
?-listing.
```

Введите запрос:

```
?-любит('Билл',бейсбол).   % Любит ли Билл бейсбол?
```

Получите ответ yes (да) и новое приглашение к запросу. Введите следующие запросы и посмотрите на результаты.

Листинг 1.3

```
?-любит('Билл', теннис).   %Любит ли Билл теннис?
?-любит(Кто, теннис). %Кто любит теннис?
?-любит('Марк',Что),любит('Эллен',Что).%Что любят Марк и
Эллен?
?-любит(Кто, Что).    %Кто что любит?
?-любит(Кто, _). %Кто любит?
```

При поиске решений в базе Пролога выдается первое решение.

Листинг 1.4

```
?-любит(Кто,теннис).
Кто = 'Эллен'
```

Если необходимо продолжить поиск в базе по этому же запросу и получить альтернативные решения, то вводится точка с запятой;

Если необходимо прервать выполнение запроса, (например, нужно набрать другой запрос), используйте клавишу b.

Не вводите имя файла с расширением без апострофов.

Если Вы хотите повторить один из предыдущих запросов, воспользуйтесь клавишами "стрелка вверх" или "стрелка вниз".

Перезагрузить, измененные во внешнем редакторе, файлы можно, ис- пользуя встроенный предикат make. Например так: ?-make.

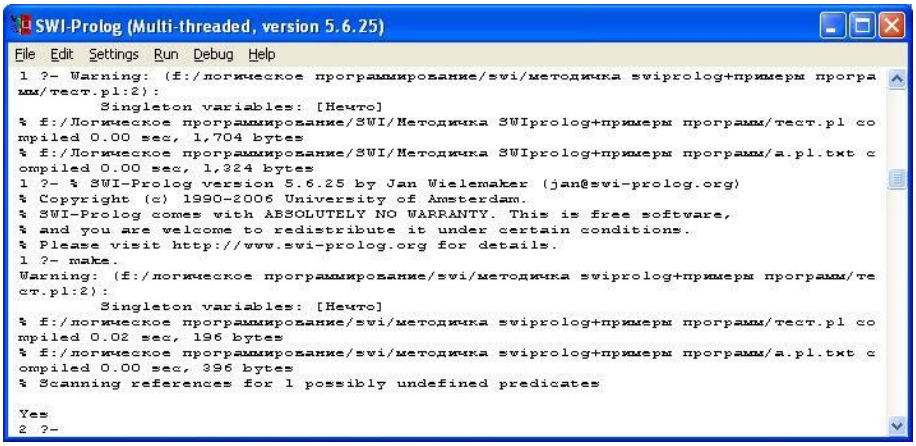

Рисунок 1.4 – встроенный предикат make

Перезагружаются все измененные файлы и файл начальной инициализации pl.ini; о его назначении будет оговорено позднее.

Использование встроенных предикатов

В интерпретаторе SWI-PROLOG имеются, как было отмечено выше, множество встроенных предикатов. Опишем некоторые из них:

consult (Имя файла).

– загрузка одного или нескольких файлов.

```
?- consult(test). test – имя файла
```

pwd

– показывает текущий рабочий каталог.

```
?- pwd. c:/prolog_workspace
```

make

– перезагрузить все измененные файлы (*.pl). В том числе и pl.ini. Аналог consult. Удобна при редактировании файлов во внешнем редакторе.

```
?- make.
```

working_directory(X,Y)

смена рабочего каталога, где X – текущий рабочий каталог, Y – новый ра- бочий каталог.

ls

просмотр списка файлов в текущем рабочем каталоге.

```
?- ls. a.pl
```

Задание 1.2

Переведите предложения русского языка в предикатную форму, создайте, сохраните и загрузите базу знаний.

Эллен любит чтение. Марк любит компьютеры. Джон любит бадминтон. Эрик любит чтение.

Бадминтон - это вид спорта. Теннис - это вид спорта.
Футбол - это вид спорта. Бейсбол - это вид спорта
Спортсмен - это тот, кто любит какой-нибудь вид спорта.
Объедините эту базу с базой из Задания 1.1.

Выполнение и трассировка программы

Алгоритм выполнения Пролог-программы основан на механизме прямого перебора с возвратом и операции сопоставления (унификации).

Выполнение программы начинается с запроса. Пролог-система берет первую подцель запроса и пытается доказать ее истинность. Для этого просматриваются программа (сверху вниз) и ищется первое утверждение, функтор и арность которого совпадают с функтором и арностью этой подцели.

Если такое утверждение существует и происходит успешное сопоставление аргументов, тогда в случае утверждения-факта - подцель доказана, и Пролог-система переходит к доказательству следующей подцели. В случае утверждения-правила Пролог-система пытается доказать истинность подцелей тела правила слева направо.

Если при поиске решения обнаружено несколько вариантов доказательства истинности цели, то Пролог-система запоминает альтернативные варианты решения - точки возврата.

Если для некоторой цели нет ни одного утверждения, с которым ее можно сопоставить, то цель считается неуспешной. Если при этом остались непройденные точки возврата, то выполняется возврат (бектрекинг) и еще раз делается попытка доказательства подцели.

Выполнение программы на языке Пролог

Листинг 1.5

```
База знаний:
?- a(X),b(X),e. a(1).
a(Y):- c(Y),d(Y). b(2).
c(1).
c(2).
d(2).
e.
Запрос
?- a(X),b(X),e. ДЕРЕВО ВЫВОДА
```

На рисунках представлено дерево вывода. Жирными линиями в дереве обозначены И-деревья – для доказательства такого дерева необходимо, чтобы каждая его ветка «опиралась» на истинное утверждение. ИЛИ-деревья исходят из вершин с точками возврата, такое дерево истинно, если хотя бы одна ветка опирается на истинное утверждение. Пунктирные линии – альтернативные ветки, не рассматриваемые на данном этапе (либо ложные, либо еще не рассмотренные).

В данном примере получение решения складывается из трех этапов. Вначале рассматривается первая подцель – она имеет два предложения в базе, заголовки которых сопоставимы с ней, поэтому она является точкой возврата. Выбирается первое сверху предложение – это факт a(1), при этом свободная переменная X получает значение 1. Первая цель запроса доказана. Вторая цель - b(1) (X уже связана со значением 1) не сопоставляется ни с одним правилом базы, следовательно, является ложной.

Бэктрекинг – возврат к ближайшей точке возврата, т.е. a(X). На втором дереве отображен вывод по второй альтернативе. Связываются две переменные X из запроса и Y

из правила для предиката a. Фактически запрос ?-a(X),b(X),e подменяется на новый набор целей ?-c(Y),d(Y),b(Y),e. Первая подцель – это c(Y) – она является новой точкой ветвления, так как унифицируется с двумя предложениями в базе. Первое предложение – факт приводит к унификации Y с 1, но следующая в наборе подцель d(1) – ложна.

Третье дерево отражает альтернативу для c(Y) с унификацией Y=2. Для этой альтернативы истинны все последующие цели из набора (они унифицируются с соответствующими фактами).

Задание 1.3

Загрузите Пролог-программу из задания 1.2.

В SWI-PROLOG есть два вида отладчиков: командный2 и графический3, в последнем трассировка выглядит более наглядно.

Для перехода в режим трассировки необходимо набрать встроенный предикат debug. Для выхода из режима трассировки – предикат nodebug4.

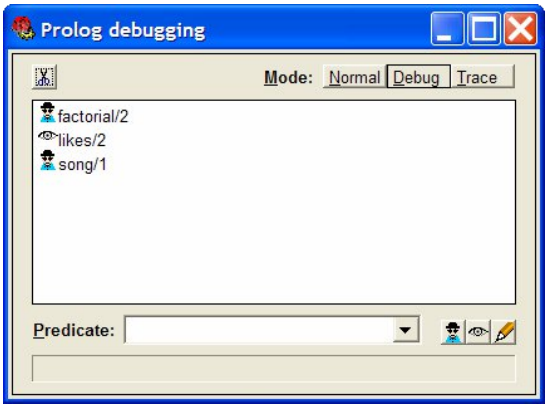

Рисунок 1.5 – Режим трассировки

Перейдите в режим трассировки. Для трассировки необходимо указать контрольные точки (Spy points) в меню Debug->Edit spy points->Predicate, где указать имена предикатов. В данном случае укажите предикат любит. Обрати- те внимание, что справа внизу находятся три кнопки, где указываются как кон- трольные точки, так и точки трассировки. Контрольные точки (Spy points) позволяют вызвать графический отладчик. Точки трассировки (Trace points)— консольный отладчик. Третий выбор — просмотр и редактирование файла программы, содержащей, введенный в поле Predicate, предикат. Внешний вид изображен на снимке ниже.

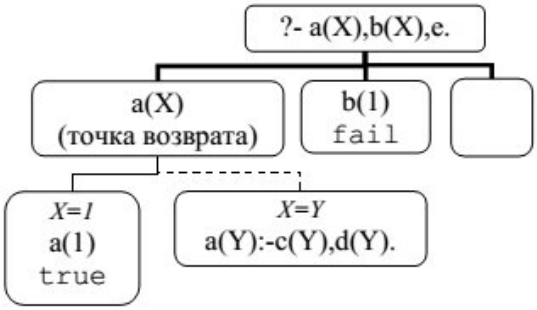

Рисунок 1.6 – Выполнение программы

Графический отладчик не поддерживал русских предикатов (не работает корректно указание контрольных точек и точек трассировки через графический интерфейс). В связи с этим опишем предикаты, относящиеся к отладке программы с использованием командной строки.

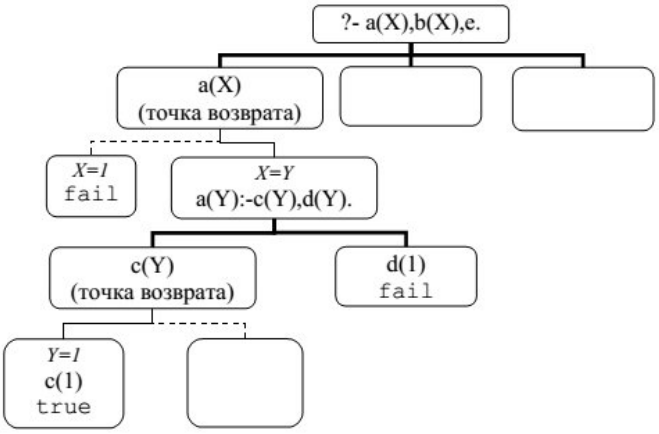

Рисунок 1.7 – Выполнение программы

Таблица 1.1 – Предикаты трассировки

Предикат	Описание
trace	Включить режим трассировки. Если не указаны точки трассировки, будет производиться полная пошаговая трассировка.
tracing	Успешен, если вызывается из режима трассировки.
notrace	Выход из режима трассировки.
guitracer, gtrace	Включить графического режима трассировки. Окно графического режима трассировки включается при встрече первой контрольной точки (spy-point).
noguitracer	Отключить графического режима трассировки.
trace(Pred)	Установить точку трассировки предиката с именем Pred (то есть отображаться при трассировке будет каждое событие связанное с предикатом Pred).
trace(Pred,Ports)	Установить точку трассировки предиката с именем Pred и активирующейся по событиям (при прохож- дении по портам) Ports. События могут быть следующих типов: fail – событие соответствующее неудачному вызову предиката с именем Pred; call – событие соответствующее первому вы- зову предиката с именем Pred; redo – событие соответствующее повторному вызову предиката с именем Pred; exit – событие соответствующее успешному окончанию вызова предиката с именем Pred. Сам же параметр Ports может принимать значения типов сообщений с префиксами соответствующими добавлению и удалению события (порта) из точки трассировки а так же списки таких значений. Например, запрос ?- trace(foo/2, +fail), trace(foo/2, [+call,-fail]). добавляет в точку трассировки связанную с преди- катом foo/2 событие fail. Для добавления события используют префикс +, для изъятия -. Кроме того можно оперировать сразу со всеми типами событий, используя обозначение all. Пример: trace(foo/2, +all).

debug	Включить режим отладки. При запросах к программе в режиме отладки выводятся все события, происходящие при выполнении запроса, связанные с установленными точками отладки.
nodebug	Отключить режим отладки.
debugging	Показать отслеживающиеся точки трассировки и контрольные точки
spy(Pred)	Установить контрольную точку, связанную с предикатом Pred.
ospy(Pred)	Убрать контрольную точку, связанную с предикатом Pred.
nospyall	Убрать все контрольные точки.

2 Отладка происходит в командной строке в диалоговом окне SWI-PROLOG и отображает последовательность выполняемых подцелей и результат выполнения.

3 Графический отладчик отображает информацию в собственном – отдельном окне.

4 Не забывайте каждый раз ставить точку после предикатов или группы предикатов (записанных через запятую) – каждое предложение заканчивается точкой.

Тогда именно на указанных предикатах трассировка будет останавливаться и показывать промежуточный результат. Остальные не указанные предикаты не будут показаны (если не указаны никакие контрольные точки, то режим трассировки будет неотличим от обычного режима работы интерпретатора).

Задайте запрос

```
?-любит(X,теннис),любит(X,компьютеры).
```

и посмотрите, что произойдет. После получения первого решения получите все остальные, вводя;. Трассировка изображена на следующем рисунке:

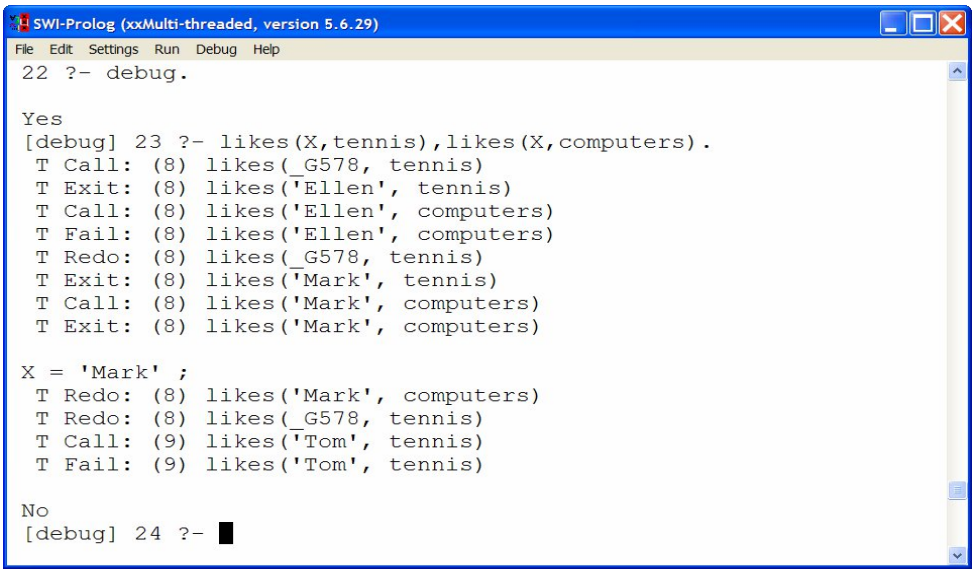

Рисунок 1.7 – Трассировка программы

В левой верхней части окна графического отладчика отображены текущие значения переменных текущего выполняемого предиката, справа стек вызовов предикатов, в нижней части факты и правила, где цветом отмечен следующий вызываемый предикат. Внешний вид графического отладчика изображен на рисунке. Сверху, в виде кнопок, находятся допустимые действия. Вторая кнопка (стрелка вправо) позволяет выполнять программу пошагово (предикат за предикатом). Цвет выделения также имеет значение. Красный цвет обозначает неудачу, ложность решения.

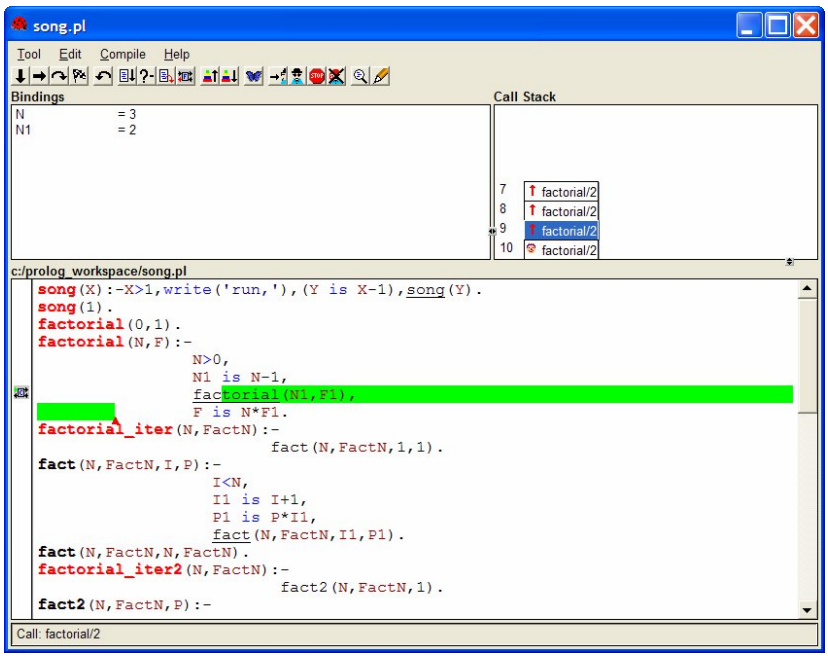

Рисунок 1.8 – Результат трассировки

3.2. СТРУКТУРА ПРОГРАММЫ НА ЯЗЫКЕ PROLOG. СИНТАКСИС ЯЗЫКА PROLOG

Синтаксис языка Пролог

При формальном описании синтаксиса конструкций алгоритмических языков часто используется так называемая «нормальная форма Бэкуса-Наура» (БНФ), разработанная в 1960 Джоном Бэкусом и Питером Науром. Впервые БНФ была применена Питером Науэром при записи синтаксиса языка Алгол-60. Основными конструкциями БНФ являются следующие.

Символ «::=» читается как «по определению» («это», «есть»). Слева от разделителя располагается объясняемое понятие, справа - конструкция, разъясняющая его. Например,

<Имя> ::= <Идентификатор>

В угловые скобки «< >» заключается часть выражения, которая используется для обозначения синтаксической конструкции языка, в частности объясняемое понятие. В приведенном выше примере это <Имя> и <Идентификатор>.

Символ «|» означает в нотации БНФ «или». Он применяется для разделения альтернативных толкований определяемого понятия. Например, десятичную цифру можно определить следующим образом:

<цифра> ::= 0|1|2|3|4|5|6|7|8|9

Часть синтаксической конструкции, заключенная в квадратные скобки, является необязательной (может присутствовать или отсутствовать), например

<Целое число> ::= [-]<Положительное целое число>

означает, что целое число можно определить через положительное целое число, перед которым может стоять знак минус.

Символ «*» обозначает, что часть синтаксической конструкции может повторяться произвольное число раз (ноль и более). Заметим, что иногда вместо символа «*» используют фигурные скобки «{ }». Например, положительное целое число в нотации БНФ можно следующим образом:

<Положительное целое число> ::= <цифра>[<цифра>]*

или

<Положительное целое число> ::= <цифра>{<цифра>}.

Т.е. положительное целое число состоит из одной или нескольких цифр.

Программа на Прологе состоит из предложений (утверждений). Каждое предложение заканчивается точкой.

Предложения бывают трех видов: факты, правила, вопросы.

Предложение имеет вид

B :-

A1, ... , An.

B называется заголовком или головой предложения, а A1, ..., An - телом. Предложение является перевернутой импликацией (B :- A эквивалентно B ¬ A, «B следует из A») или фразой Хорна. Символ «:-» означает «следует из», символ «,» – конъюнкция (логическое И, Ù).

При необходимости применения дизъюнкции (логическое ИЛИ, Ú) используется символ «;», действующий до следующей дизъюнкции, окончания правила или закрывающей круглой скобки. Например,

Пролог	Логика предикатов
C :- A1, ... , An; B1, ... , Bm.	(A1 Ù ... Ù An) Ú (B1 Ù ... Ù Bm) ® C
или	
D :- A, (B ; C).	A Ù (B Ú C) ® D

Факт фиксирует (определяет) некоторое отношение между объектами. Он состоит только из заголовка. Можно считать, что факт – это предложение, у которого нет тела. В терминах логики предикатов, факт – это и есть предикат.

Например, факт, что Наташа является мамой Даши, может быть записан в виде (в SWI-Prolog строки-константы записываются в одинарных кавычках):

mama('Наташа', 'Даша').

Факт представляет собой безусловно истинное утверждение.

Если воспользоваться нормальной формой Бэкуса-Науэра, то предикат можно определить следующим образом:

<Предикат> ::= <Имя> | <Имя>(<аргумент> [,<аргумент>]*),

т.е. предикат состоит либо только из имени, либо из имени и следующей за ним последовательности аргументов (термов), заключенной в скобки.

Аргументом (термом) предиката может быть константа, переменная или составной объект (список или функция). Число аргументов предиката называется арностью.

Правило – предложение, истинность которого зависит от истинности одного или нескольких предложений. Обычно правило содержит несколько хвостовых целей, которые должны быть истинными для того, чтобы само правило было истинным.

В нотации БНФ правило будет иметь вид:

<Правило> ::= <предикат> :- <предикат> [,<предикат>]*.

Например. Известно, что бабушка человека - это мама его мамы или мама его папы. Соответствующие правила будут иметь вид:

```
grandmama(X, Y):-
mama(X, Z), mama(Z, Y).
grandmama(X, Y):-
mama(X, Z), papa(Z, Y).
или
grandmama(X, Y):-
mama(X, Z), mama(Z, Y);
mama(X, Z), papa(Z, Y).
```

или

```
grandmama(X, Y):-
mama(X, Z), (mama(Z, Y); papa(Z, Y)).
```

Имя переменной в Прологе может состоять из букв латинского алфавита, цифр, знаков подчеркивания и должно начинаться с прописной буквы или знака подчеркивания. При этом переменные в теле правила неявно связаны квантором всеобщности и эквивалентны объектам предметной области.

Переменные могут быть свободными или связанными.

Свободная переменная – переменная, которая еще не получила значения. Она не равняется ни нулю, ни пробелу; у нее вообще нет никакого значения. Такие переменные еще называют неконкретизированными.

Переменная, которая получила какое-то значение и оказалась связанной с определенным объектом, называется связанной. Если переменная была конкретизирована каким-то значением и ей сопоставлен некоторый объект, то эта переменная уже не может быть изменена.

Областью действия переменной в Прологе является одно предложение. В разных предложениях может использоваться одно и то же имя переменной для обозначения разных объектов. Исключением из правила определения области действия является анонимная переменная, которая обозначается символом подчеркивания «_». Анонимная переменная предписывает интерпретатору (компилятору) проигнорировать значение аргумента (терма). Если в правиле несколько анонимных переменных, то все они отличаются друг от друга, несмотря на то, что записаны с использованием одного и того же символа («_»). Анонимные переменные могут записываться только в качестве терма предиката. Использовать их в выражениях (например, арифметических) нельзя.

Третьим специфическим видом предложений Пролога можно считать вопросы (запросы, цели).

Вопрос состоит только из тела и может быть выражен с помощью БНФ в виде:

<Вопрос> ::= <Предикат> [,<Предикат>]*

Вопросы используют для выяснения выполнимости некоторого отношения между описанными в программе объектами. Система рассматривает вопрос как цель, к которой надо стремиться. Ответ на вопрос может оказаться положительным (true) или отрицательным (false), в зависимости от того, может ли быть достигнута соответствующая цель. Программа может содержать вопрос в теле (внутренняя цель). Если программа содержит внутреннюю цель, то после запуска программы на выполнение система сразу проверяет достижимость заданной цели. Если внутренней цели в программе нет, то после запуска программы система выдает приглашение вводить вопросы в диалоговом режиме (внешняя цель). Программа, компилируемая в исполняемый файл, обязательно должна иметь внутреннюю цель.

Если цель достигнута, система отвечает «yes» («true»), в противном случае «no» («false»). Следует отметить, что ответ «no» на вопрос не всегда означает, что он отрицательный. Система может дать такой ответ и в том случае, когда у нее просто недостаточно информации, позволяющей положительно ответить на вопрос. Т.е. Пролог основан на т.н. «модели закрытого мира», в которой все, что можно получить на основе описания модели является истиной, а остальное – ложью.

Факт, правило и вопрос с точки зрения фраз Хорна (B ¬ A) можно интерпретировать следующим образом:

- факт: B ¬ true;
- правило: B ¬ A;
- вопрос: true ¬ A.

Т.о. программа (база знаний) на Прологе состоит из фактов и правил, которые представляют собой продукции с предикатами в левой и правой части. Тем самым

программа выражает некоторые знания о предметной области, необходимые для решения задачи. Запрос - это также некоторый предикат, истинность которого нас интересует. Если запрос не содержит переменных, то вычисление его значения дает ответ «true» при его истинности, либо ответ «false» при его ложности. Если же в предикате запроса есть переменные, то ищутся их значения (интерпретация), при которых этот предикат и все предикаты программы становятся истинными. В этом и состоит суть вычислений (логического вывода) программы на Прологе.

Пример программирования на языке Пролог

Рассмотрим несколько примеров. Пусть в программе заданы следующие факты (предикаты) и правила:

Листинг 2.1

```
mama('Наташа', 'Даша').
mama('Даша', 'Маша').
mama('Наташа', 'Вася').

papa('Вася', 'Маша').

grandmama(X, Y):-
mama(X, Z), (mama(Z, Y); papa(Z, Y)).

grandpapa(X, Y):-
papa(X, Z), (mama(Z, Y); papa(Z, Y)).
```

Вопрос 1 – является ли Наташа мамой Даши:

```
?- mama('Наташа', 'Даша').
true.
```

Вопрос 2 – кто является мамой Даши:

```
?- mama(X, 'Даша').
X = 'Наташа' ;
false.
```

Первая строка сообщения означает, что ответ найдет и мамой Даши является Наташа; вторая – что в базе знаний для оставшихся предложений не обнаружены другие мамы Даши.

Вопрос 3 – есть ли у Даши мама:

```
?- mama(_, 'Даша').
true.
```

Вопрос 4 – найти всех мам и детей:

```
?- mama(X,Y).
X = 'Наташа',
Y = 'Даша' ;
```

```
X = 'Даша',
Y = 'Маша' ;
X = 'Наташа',
Y = 'Вася'.
```

В данном случае после третьего ответа не выдается «false», т.к. в базе знаний были перебраны все предложения и они все истинны.

Вопрос 5 – для кого Наташа является бабушкой:

```
?- grandmama('Наташа', X).
X = 'Маша' ;
X = 'Маша'.
```

В данном случае выдается два одинаковых ответа «Маша», т.к. правило grandmama в одном случае сработало по цепочке «Наташа – Даша – Маша», а в другом - «Наташа – Вася – Маша». Очевидно, что в приведенном примере базы знаний либо Наташи - это две различные женщины, либо Маши.

Краткие сведения об операциях и встроенных предикатах SWI-Prolog

В табл. 2.1 приведены некоторые операции и предикаты SWI-Prolog, которые в дальнейшем будут использоваться для иллюстрации примеров.

Таблица 2.1 Некоторые операции и предикаты SWI-Prolog

Операция / Предикат	Назначение
true	Истина
fail, false	Ложь
=	Унификация (присваивание значения несвязанной переменной)
<, =<, >=, >	Арифметические (только для чисел) операции сравнения
=:=	Арифметическое равенство
=\=	Арифметическое неравенство
is	Вычисление арифметического выражения (например, A is 5 + 2)
@<, @=<, @>=, @>	Операции сравнения для констант и переменных любого типа (чисел, строк, списков и т.д.)
==	Равенство констант и переменных любого типа
\==	Неравенство констант и переменных любого типа
not(A)	Отрицание логического выражения A
read(A)	Чтение значения с клавиатуры и присваивание его переменной A
write(A)	Печать A на экран с установкой курсора после последнего напечатанного символа
writeln(A)	Печать A на экран с переводом курсора в начало следующей строки
nl	Перевод курсора в начало следующей строки
repeat	Предикат, выдающий новое истинное значение при возврате. Передоказываемый предикат
!	Предикат (cut, сократить), запрещающий возврат далее той точки, где он стоит
assert(A), assertz(A)	Динамическое добавление факта (правила) в конец списка подобных фактов (правил) базы знаний (программы)
asserta(A)	Динамическое добавление факта (правила) в начало списка подобных фактов (правил) базы знаний
retract(A)	Удаление первого факта (правила) базы знаний
retractall(A)	Удаление всех фактов (правил) базы знаний с именем A

Процедура вывода в Прологе

При поиске решения (доказательства цели) в Прологе используется метод перебора с возвратами (поиск в глубину). Пролог при доказательстве утверждения поочередно пытается установить истинность входящих в него предикатов (утверждений). Если первый предикат истинен, то Пролог переходит ко второму. Если и он истинен, то переходит к третьему. Если второй предикат ложен, то Пролог пытается установить его истинность при других значениях, входящих в него переменных. Если этого не удается сделать, то он возвращается к первому предикату и пытается установить его истинность для новых значений переменных, а затем снова возвращается к доказательству второго предиката. Такая процедура повторяется до тех пор, пока не будет достигнута истинность последнего предиката. После доказательства истинности последнего предиката цели Пролог завершает работу. Процесс возврата в Прологе называется backtracking.

Например, пусть имеется запрос на определения внучек и внуков «Наташи»:

```
?- mama('Наташа', Y),
(mama(Y, Z); papa(Y, Z)),
write(Z);
write('Всё').
Ответ.
Маша;
Маша;
Всё;
```

Выделенная жирным часть соответствует правилу определения бабушки (grandmama).

3.3 АЛГОРИТМЫ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ. РЕШЕНИЕ ЗАДАЧ НА РЕКУРСИЮ

Рекурсия в Прологе

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

Пусть имеются следующие факты о том, какая валюта котируется выше: doroje(dollar, rubl).

Листинг 3.1

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

Выполним запрос:

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

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

```
? doroje(evro, iena).
```

То ответ будет отрицательный, поскольку такой факт отсутствует.

Аналогичным будет ответ на вопрос:

```
? doroje(funt, rubl).
```

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

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

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

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

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

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

Листинг 3.2

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

Первое предложение в этой конструкции определяет момент прекращения рекурсивных вызовов.

Второе правило описывает возможности рекурсивных вызовов, когда существуют непроверенные варианты решения. Вообще, любая рекурсивная процедура должна содержать:

Нерекурсивное правило, применяемое для завершения рекурсии (их может быть несколько, задающие разные условия для завершения рекурсии),

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

Набор правил просматривается сверху вниз. Сначала делается попытка выполнения нерекурсивного правила. Если оно отсутствует, то рекурсивное правило может работать бесконечно.

Применение рекурсии в логическом программировании

Рекурсия – процесс повторения элементов самоподобным образом. **Рекурсия** (в программировании) – алгоритмический метод, заключающийся в возможности обращения правила (функции, процедуры) к самому себе один или более раз.

Рекурсия является часто используемым приемом в программах на Прологе. Для рассмотренного выше примера базы знаний «Родственники» введем правило, описывающее отношение «предок» - «потомок» с помощью рекурсии.

Листинг 3.3 Родственники

```
% 1 параметр - имя отца или матери
% 2 параметр - имя сына или дочери
roditel('Вася', 'Вика').
roditel('Вася', 'Коля').
roditel('Коля', 'Юля').
```

```
roditel('Юля', 'Миша').
roditel('Юля', 'Маша').
roditel('Коля', 'Света').
roditel('Света', 'Рома').
roditel('Света', 'Леша').
% 1 параметр - предок
% 2 параметр - потомок
predok(A, B) :- roditel(A, B).
predok(A, B) :- roditel(A, C), predok(C, B).
```

На вопрос predok('Вася', 'Миша') ответ будет положительным.

Второй пример использования рекурсии – расчет факториала.

Листинг 3.4 Расчет факториала

```
fact(1, 1):- !. % факториал единицы равен единице
fact(N, F):-
N1 is N - 1,
fact(N1, F1), % F1 равен факториалу числа на единицу
меньшего N
F is F1 * N. % факториал числа N равен произведению F1 на
само число N
```

На вопрос fact(6, F) ответ будет «F = 720».

Любая рекурсивная процедура в Прологе должна включать, как минимум два правила:

1) нерекурсивное правило, определяющее его вид в момент прекращения рекурсии;

2) рекурсивное правило. Первая подцель, вырабатывает новые значения аргументов, а вторая - вызов самого правила с новыми значениями аргументов.

Управление процессом вывода

В Прологе имеется два стандартных предиката, которые позволяют управлять процедурой перебора с возвратами:

- fail (false) – предикат, который всегда возвращает ложь;
- ! – предикат (cut, сократить), запрещающий возврат далее той точки, где он стоит.

Первый из них используется для организации циклов, а второй для ускорения процедуры перебора.

Если мы желаем узнать всех предков «Миши» и при этом, чтобы выдача предков на консоль была выполнена автоматически, а не в диалоговом режиме (путем нажатия «;» после каждого ответа), тогда запрос должен выглядеть следующим образом:

Листинг 3.5

```
?- writeln('Предки Миши:'),
predok(A, 'Миша'),
writeln(A),
fail;
true.
```

```
Ответ:
Предки Миши:
Юля
Вася
Коля
```

Таким образом, предикат fail выполняет принудительный откат и заставляет Пролог заново передоказывать все предыдущие подцели с новыми значениями переменных. Передоказательству в приведенном примере будет подвержен только предикат predok(A, 'Миша'), т.к. предикаты write и writeln не генерируют новые значения переменных. Если после предиката fail находятся другие предикаты правила, указываемые через «,» (логическое И), то они никогда не будут выполнены.

Рассмотрим более сложный пример: выяснить всех правнуков «Коли».

Листинг 3.6

```
?- roditel('Коля', A),
write('Правнуки ('), write(A), writeln('):'),
roditel(A, B),
writeln(B),
fail;
true.
Ответ:
Правнуки (Юля):
Миша
Маша
Правнуки (Света):
Рома
Леша
```

Если в данной цели использовать предикат ! после roditel('Коля', A), то результатом работы программы будет первых три строки, если после roditel(A, B) – первых две строки. Если при выполнении правила в результате возврата для передоказательства предикатов с новыми значениями переменных будет встречен предикат !, то всё правило в целом возвратит «false», даже если у него имеются альтернативные пути доказательства, указываемые через «;» (логическое ИЛИ) или определения, задаваемые отдельными предложениями.

3.4 ПРИМЕНЕНИЕ СПИСКОВ В ПРОЛОГЕ

Использование списков

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

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

Элементами списка могут быть любые термы Пролога, т.е. атомы, числа, переменные и составные термы. Каждый непустой список может быть разделен на голову – первый элемент списка и хвост – остальные элементы списка. При этом список представляется в виде:

```
[A|B] или [1| [2,3]] или [1| B] или [“вчера”|
[“сегодня”,”завтра”]].
```

Это позволяет всякий список представить в виде бинарного дерева (рис.1). У списка, состоящего только из одного элемента, головой является этот элемент, а хвостом – пустой список. Для использования списка необходимо описать предикат списка.

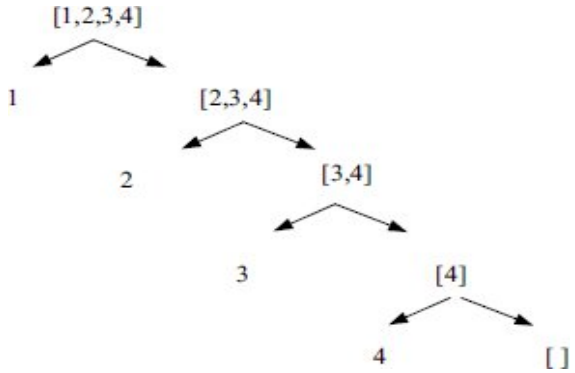

Рисунок 4.1 Бинарное дерево списка.

В следующем примере 1 - голова списка, а [2, 3, 4, 5] - хвост. Пролог покажет это при помощи сопоставления списка чисел с образцом, состоящим из головы и хвоста.

```
?- [1, 2, 3, 4, 5] = [Head | Tail].

Head = 1

Tail = [2, 3, 4, 5]

Yes
```

Здесь Head и Tail - только имена переменных. Мы могли бы использовать X и Y или какие-нибудь другие имена переменных с тем же успехом. Заметим, что хвост списка всегда является списком. Голова, в свою очередь, есть элемент списка, что верно и для всех других элементов, расположенных до вертикальной черты. Это позволяет получить, скажем, второй элемент списка.

В Prolog используется специальный символ для разделения списка на голову и хвост – вертикальная черта |.

Например:

```
[1, 2, 3] или [1 | [2, 3]] или [1 | [2| [3]]] или [1 | [2
| [3 | [ ]]]]
```

Вертикальную черту можно использовать не только для отделения головы списка, но и для отделения произвольного числа начальных элементов списка:

```
[1, 2, 3] или [1, 2 | [3]] или [1, 2, 3 | [ ]]
```

Пример

Используем анонимные переменные для головы и списка, стоящего после черты, если нам нужен только второй элемент списка:

```
?- [слон, лошадь, осел, собака] = [_, X | _ ]. X = лошадь

Yes
```

Рассмотрим несколько процедур обработки списков. Обратите внимание, что все они используют рекурсию, в которой терминальное (базовое) правило определено для пустого списка.

Пример

Предикат «перестановка» выдает списки, полученные перестановкой элементов своего первого аргумента.

перестановка([],[]).

перестановка([H|L],Z):- перестановка(L,Y), место(H,Y,Z).

Пример использования:

Листинг 4.1

```
?- перестановка([a,b,c],X). X = [a, b, c] ;
X = [b, a, c] ;
X = [b, c, a] ;
X = [a, c, b] ;
X = [c, a, b] ;
X = [c, b, a] ; No
```

Предположим, что имеется некоторый список, в котором X обозначает его голову, а Y — хвост списка. Такой список можно записать, как [X|Y]. Этот список может содержать, например, инструмент для обработки деталей:

[фреза, сверло, резец, хон]

Теперь предположим, что мы хотим определить, содержится ли некоторый инструмент в указанном списке. В Прологе это можно сделать, определив, совпадает ли данный инструмент с головой списка. Если совпадает, то наш список завершается успехом. Если нет, то мы проверяем, есть ли нужный инструмент в хвосте исходного списка. Это значит, что снова проверяется голова, но уже очередного хвоста списка. Если мы доходим до конца списка, который будет пустым списком, то наш поиск завершается неудачей: указанного инструмента в исходном списке нет.

Для того, чтобы записать все это на Прологе, сначала надо:

Установить, что между объектом и списком, в который этот объект может входить, существует отношение. Для записи этого отношения будем использовать предикат принадлежит: целевое утверждение принадлежит(X,Y) является истинным («выполняется»), если терм, связанный с X, является элементом списка, связанного с Y. Имеются два условия, которые надо проверить для определения истинности предиката. Первое условие говорит, что X будет элементом списка Y, если X совпадает с головой списка Y. На Прологе этот факт записывается следующим образом:

```
принадлежит(X,[X|_]).
```

Эта запись констатирует, что X является элементом списка, который имеет X в качестве головы. В данном случае анонимная переменная «_» для обозначения хвоста списка.

Второе правило говорит о том, что X принадлежит списку при условии, что он входит в хвост этого списка, обозначаемый через Y. Для этого используем тот же самый предикат принадлежит для того, чтобы определить, принадлежит ли X хвосту списка. В этом и состоит суть рекурсии. На Прологе это выглядит так:

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

Два этих правила в совокупности определяют предикат для отношения принадлежности и указывают Прологу, каким образом просматривать список от начала до конца при поиске некоторого элемента в списке. Наиболее важный момент, о котором следует помнить, встретившись с рекурсивно определенным предикатом, заключается в том, что прежде всего надо найти граничные условия и способ использования рекурсии.

Для предиката «принадлежит» в действительности имеются два типа граничных условий. Либо объект, который мы ищем, содержится в списке, либо нет. Первое граничное условие для предиката «принадлежит» распознается первым утверждением, которое приводит к прекращению поиска в списке, если первый аргумент предиката принадлежит совпадает с головой списка. Второе граничное условие встречается, когда второй аргумент предиката «принадлежит» является пустым списком.

Это все демонстрирует следующий пример на Прологе: принадлежит(X,[X|_]).

```
принадлежит(X,[_|Y]):- принадлежит(X,Y).
?- принадлежит(фреза,[фреза, сверло, резец, хон]).
true
?- принадлежит(протяжка,[фреза, сверло, резец, хон]).
false.
```

Способы организации циклов

В Прологе отсутствуют конструкции циклов с параметром, пред- и постусловием, но с помощью соответствующих механизмов можно организовать разные типы циклов.

1 способ. Используя рекурсию.

2 способ. Используя предикат, который можно передоказать, и предикат fail.

3 способ. Используя предикат, который можно передоказать, и логическое утверждение. Например, выдать пары «предок» - «потомок» до тех пор, пока не встретится потомок с именем «Миша».

Листинг 4.2

```
?- predok(A, B),
write(A), write(' является предком '), writeln(B),
B == 'Миша';
true.
Ответ:
Вика является предком Коля
Вася является предком Коля
Коля является предком Юля
Юля является предком Миша
```

Такая конструкция соответствует циклу с постусловием.

4 способ. Организация цикла со счетчиком, используя предикат repeat и динамическое добавление фактов в базу знаний (программу).

Следует отметить, что в некоторых реализациях языка Пролог отсутствует встроенный предикат repeat. Тогда данный предикат надо определить в программе следующим образом

repeat.

repeat:- repeat.

Его используют вместо передоказываемого предиката. Но если при передоказательстве обычных предикатов может наступить момент, что все факты

исчерпаны (например, predok(A, ‘Миша’) ... fail), то при использовании предиката repeat такой момент не наступает никогда.

Листинг 4.3

```
?- assert(count(1)),
repeat,
retract(count(X)),
X2 is X * X,
write(X), write('^2 = '), writeln(X2),
X1 is X + 1,
(X1 > 10, !;
assert(count(X1)), fail).
```

Приведенная программа будет циклически выдавать на экран квадраты чисел от 1 до 10. В качестве счетчика используется предикат (факт) count, который динамически добавляется и удаляется из базы знаний.

Существуют также другие способы организации циклов.

Применение списков в Прологе

В Прологе нет такой распространенной и часто используемой структуры хранения данных как массивы, но зато есть развитые возможности работы со списками. Список – упорядоченный набор элементов одного типа. В отличие от массивов, количество элементов которых строго фиксировано (в большинстве языков программирования), списки позволяют модифицировать, добавлять или удалять из него элементы.

Списки в Прологе заключаются в квадратные скобки, например [1, 2, 8, 123] или ['Пн', 'Вт', 'Четверг']. Список, не содержащий ни одного элемента «[]», называется пустым. Каждый непустой список состоит из двух частей: головы и хвоста. Головой является первый элемент списка, хвостом – все остальное.

Таблица 4.1 Списки и его составные части

Список	Голова	Хвост
[1, 2, 8, 123]	1	[2, 8, 123]
['Пн', 'Вт', 'Четверг']	'Пн'	['Вт', 'Четверг']
[1.25]	1.25	[]
[]	не определена	не определен

В программе голова отделяется от хвоста символом «|».

Часто используемыми операциями при работе со списками являются:

- проверка наличия элемента в списке;
- добавление элемента в список;
- конкатенция (объединение) списков;
- удаление элемента из списка;
- задание обратного порядка следования элементов списка;
- разделение списка на два.

1 Проверка наличия элемента в списке.

Листинг 4.4

```
% 1 параметр – элемент, наличие которого в списке
требуется проверить
```

```
% 2 параметр - список
member(X, [X |_]).
member(X, [_|Tail]):- member(X, Tail).

?- member(2, [1, 3, 45, 2, 74]),
write('Да');
write('Нет').
Ответ: Да.
```

2. Добавление элемента в список. Для данной операции не требуется отдельного правила, если элемент X добавляется в начало списка List

```
NewList = [X| List].
```

3. Конкатенция двух списков.

Листинг 4.5

```
% 1 параметр – первый список
% 2 параметр – второй список
% 3 параметр – результат объединения двух списков
concat([], L2, L2).
concat([X|L1], L2, [X|L3]):- concat(L1, L2, L3).

?- concat([1, 2], [3, 45], L),
write(L).
Ответ: [1, 2, 3, 45].
```

4. Удаление элемента из списка и задание обратного порядка следования элементов списка.

Листинг 4.6

```
% 1 параметр – удаляемый элемент
% 2 параметр – исходный список
% 3 параметр – рабочий список
% 4 параметр – перевернутый список без элемента
delete(_, [], L, L).
delete(X, [X|L], L1, L2):- delete(X, L, L1, L2).
delete(X, [Y|L], L1, L2):- X \== Y, delete(X, L, [Y|L1],
L2).

% 1 параметр – исходный список
% 2 параметр – рабочий список
% 3 параметр – перевернутый список
reverse([], Lr, Lr).
```

```
reverse([X|L], L1, Lr):- reverse(L, [X|L1], Lr).


?- delete(1, [1, 3, 1, 45, 1], [], L),
reverse(L, Lr),
write(Lr).
Ответ: [3, 45].
```

5. Разделение списка на два.

Листинг 4.7

```
% 1 параметр – элемент, задающий разбиение
% 2 параметр – исходный список
% 3 параметр – элементы, меньшие или равные 1 параметру
% 4 параметр – элементы, большие 1 параметра
split(_, [], [], []).
split(Y, [X|L], [X|L1], L2):- X @<= Y, split(Y, L, L1,
L2).
split(Y, [X|L], L1, [X|L2]):- X @> Y, split(Y, L, L1,
L2).


?- split(7, [1, 3, 1, 45, 1, 33], L1, L2),
writeln(L1),
writeln(L2).
Ответ: [1, 3, 1, 1] и [45, 33].
```

3.5. РАЗРАБОТКА ЭКСПЕРТНЫХ СИСТЕМ

Разработка экспертных систем на языке Пролог

После запуска интерпретатор Пролога выдает приглашение вводить *запросы*, которые также называют *вопросами* или *целями*. Как правило, запрос представляет собой вызов процедуры и записывается как имя процедуры, за которым в круглых скобках следуют аргументы, разделенные запятыми. В конце запроса ставится точка. Процедуры в Прологе называют *предикатами*. Некоторые из них встроены в язык и их можно вызывать немедленно.

```
?- true.


Yes


?- fail.


No
```

Встроенные предикаты `true` и `fail` представляют соответственно тождественно истинное и тождественно ложное высказывание. Ответ на первый вопрос всегда "да" , а на

второй "нет". Говорят, что запрос `true` заканчивается *успехом*, а `fail` - *неудачей*. Точно так же любой запрос завершается либо успехом, либо неудачей.

Листинг 5.1

```
?- integer(1).

Yes

?- 3+4 < 3*2.

No

?- write('привет').
привет

Yes
```

Обратите внимание: между именем предиката и открывающей скобкой не должно быть пробелов. Второй запрос записан в традиционной инфиксной форме - некоторые предикаты допускают такую запись. Выполнив последний запрос, Пролог отвечает Yes, хотя мы вроде бы ни о чем не спрашивали. Таким образом он просто сообщает о благополучном завершении запроса.

Есть еще одна возможность - запрос может завершиться ошибкой.

Листинг 5.2

```
?- abracadabra.
ERROR: Undefined procedure: abracadabra/0

?- 3/0 > 1.
ERROR: //2: Arithmetic: evaluation error: `zero_divisor'
```

Поскольку системе не известно ничего по поводу `abracadabra`, возникает исключительная ситуация. Во втором случае ошибка возникла в процессе арифметических вычислений. В принципе, ошибка - тоже неудача, но неудача особого рода, после которой дальнейшее выполнение запроса не имеет смысла.

Имея некоторый опыт работы с функциональными языками, мы можем попытаться найти сумму чисел:

```
?- 2 + 3.
ERROR: Undefined procedure: (+)/2
```

и получаем сообщение об ошибке. Наш запрос не имеет формы предложения. Попробуем по другому.

```
?- write(2+3).
```

```
2+3
Yes
```

Система выводит выражение, даже не пытаясь что-нибудь вычислить. Может быть так?

```
?- X = 2+3.

X = 2+3
Yes
```

Снова неудача. Пролог разрабатывался прежде всего как язык обработки символьной информации и выражения вида "2+3" - это его объекты данных (они называются *термами*). Именно это выражение и присваивается переменной X. Но надо же как то выполнять арифметические вычисления. Для этого предназначены специальные *арифметические предикаты*.

Листинг 5.3

```
?- X is 2+3.

X = 5
Yes
```

Наконец-то! Предикат `is` вычисляет значение выражения и присваивает его переменной. Запрос завершается успехом и значение переменной выводится в качестве ответа.

Другие арифметические предикаты (`=:=`, `=\=`, `<`, `=<`, `>`, `>=`) вычисляют оба своих аргумента и сравнивают их значения.

Листинг 5.4

```
?-  3 + 2 =:= 5.

Yes

?- 2 * 2 =:= 2 + 2 .

Yes

?- exp(2) >= sin(2) .

Yes
```

Другая разновидность примитивных объектов - *атомы*. Чаще всего они записываются как последовательности букв и цифр, начинающейся со *строчной* буквы. Это отличает их от переменных, которые начинаются с *прописной* буквы. Атомы можно сравнивать посредством предикатов `==` , `\==`, `@<`, `@=<`, `@>`, `@>=` . Арифметические предикаты для этого не подходят, поскольку они попытаются вычислить арифметические

значения атомов, что приведет к ошибке. А вот числа (но не значения выражений) можно сравнивать и теми и другими.

Листинг 5.5

```
?- a @< b.
Yes

?- 1 @< 2.
Yes

?- 2 < a.
ERROR: Arithmetic: `a/0' is not a function

?- 2 < e.
Yes
```

Почему последний запрос успешен, догадайтесь сами.

```
?- X is 2+3.

X = 5
Yes

?- Y is X+1.
ERROR: Arguments are not sufficiently instantiated
```

В сообщение об ошибке указано что переменная(имеется в виду X) не конкретизирована. Но мы же только что присвоили ей значение! Дело в том, что это уже другая переменная. Область действия переменной распространяется только на один запрос. Но запросы могут быть и сложными, состоящими из нескольких целей.

Листинг 5.6

```
?- X is 2+3, Y is X+1.

X = 5
Y = 6

Yes
```

Пока наша возможность задавать вопросы ограничена. Чтобы задавать более интересные вопросы, надо загрузить какую-нибудь программу. Это можно процедурой `consult`, для которой придумано сокращение. Например, если программа находиться в файле 'my_program.pl', то ее можно загрузить, введя

```
?- consult(my_program).
```

или

```
?- [my_program].
```

А команда

```
?- [my_program,another_program,yet_another_program].
```

загрузит сразу все указанные программы. Но прежде чем загружать программу надо ее написать.

Программирование базы знаний на языке Пролог

Программы на Прологе часто называют "базами данных" или даже "базами знаний" в том смысле, что они представляет собой совокупности предложений, определяющих отношения между объектами предметной области или свойства этих объектов. Свойства и отношения в Прологе, как и в логике, называют предикатами.

Каждый предикат определяется последовательностью *предложений*, которую называют *процедурой*. Традиционно выделяют два типа предложений: *факты* и *правила*. Для начала разберемся с первой разновидностью. Предложение-факт состоит из имени предиката, за которым в круглых скобках следуют аргументы. Такое предложение описывает утверждение о конкретных объектах. Программа, состоящая только из фактов в сущности является реляционной базой данных.

Возьмем в качестве примера фрагмент базы данных небольшой компании Horns&Hoofs Ltd. Одноместный предикат `employee` устанавливает свойство "быть работником".

Листинг 5.7.1

```
employee(anthony).
employee(barbara).
employee(charles).
employee(diana).
employee(edward).
employee(frederic).
employee(gregory).
employee(herbert).
employee(isabella).
employee(john).
```

Вообще то, имена собственные положено писать с прописной буквы, но в Прологе имена отношений и объектов должны начинаться со строчной буквы или заключаться в кавычки. Как правило, предпочитают обходится без кавычек. Программа может содержать предикаты с одинаковыми именами, но с разным количеством аргументов. Чтобы различать их употребляют имена состоящие из имени предиката и его арности. Полное имя только что определенного предиката: `employee/1`.

Предикат `salary/2` устанавливает оклад каждого работника.

Листинг 5.7.2

```
salary(anthony,500).
salary(barbara,200).
salary(charles,180).
salary(diana, 150).
```

```
salary(edward,150).
salary(frederic,140).
salary(gregory,150).
salary(herbert,100).
salary(isabella,90).
salary(john,160).
```

Предикат `boss/2` устанавливает описывает организационную структуру компании. Утверждение boss(A,B) означает, что B - руководитель A.

Листинг 5.7.3

```
boss(barbara, anthony).
boss(charles, anthony).
boss(diana,   barbara).
boss(edward,  barbara).
boss(frederic,charles).
boss(gregory, charles).
boss(herbert, charles).
boss(isabella,gregory).
boss(john,    gregory).
```

Загрузив программу, можно задавать вопросы, непосредственно к ней относящиеся.

Листинг 5.7.4

```
?- [hh].
% hh compiled 0.00 sec, 0 bytes
Yes

?- employee(anthony).
Yes

?- employee(ronald).
No

?- boss(barbara, anthony).
Yes
```

Это простейший тип вопросов. Система просто проверяет записано ли некоторое предложение в программе и отвечает "да" если это так или "нет" в противном случае. Таким образом, предложение считается ложным, если явно не записано обратного.

Более интересные вопросы получаются, если подставить в запросы переменные. Например, мы можем узнать зарплату Чарльза, спросив.

```
?- salary(charles,S).
```

```
S = 180
Yes
```

Отвечая на этот вопрос Пролог сопоставляет его со всеми фактами из базы данных, пытаясь подобрать подходящее значение переменной. Некоторые вопросы допускают несколько вариантов ответа. Такие вопросы, называют *недетерминированными*.

```
?- boss(A,charles).

A = frederic
Yes
```

Но это не единственный ответ. Отвечая на вопросы с переменными система, выдав значение ждет нажатия на Enter. На самом деле, она интересуется нужны ли нам другие ответы. Нажимая на Enter мы говорим "спасибо, достаточно" и система отвечает *Yes*. Чтобы получить другие ответы надо ввести ";".

```
?- boss(A,charles).

A = frederic ;
A = gregory ;
A = herbert ;
No
```

Последнее *No* означает, что других ответов нет. Ответы выдаются в том же порядке, в котором они встречаются в программе. Вопрос с двумя переменными вернет все пары подходящие значений.

```
?- boss(A,B).

A = barbara
B = anthony ;

A = charles
B = anthony ;

A = diana
B = barbara

Yes
```

В тех случаях, когда требуется указать переменную но её значение для нас несущественно можно использовать так называемую *анонимную переменную*, состоящую из единственного символа подчеркивания. Отвечая на вопрос, содержащий анонимную переменную, Пролог подбирает для нее значение, но не выводит его.

```
?- boss(A,_).
```

```
A = barbara ;
A = charles ;
A = diana ;
A = edward
Yes
```

До сих пор мы задавали довольно простые вопросы. Чтобы ответить на них достаточно было просмотреть одно отношение. Но как мы знаем, простые вопросы можно объединять в более сложные. Запятая при этом получает логический смысл конъюнкции предложений. Например, спросим не получает ли кто то больше своего начальника

```
?- boss(A,B), salary(A,SA), salary(B,SB), SA > SB.

A = john
B = gregory
SA = 160
SB = 150 ;

No
```

и обнаружим непорядок: оклад Джона больше, чем у Грегори.

Кроме конъюнкции для соединения предложений можно применять и дизъюнкцию, которая обозначается точкой с запятой ";".

```
?- salary(X,S),(S<100;S>200).

X = anthony
S = 500 ;

X = isabella
S = 90 ;

No
```

Отрицание обозначается "\+" (или `not`, но эта форма, хотя и привычная, не рекомендуется).

```
?-  \+ employee(X).

No
```

Запрос означает "не существует ни одного работника", что, конечно, ложно.

```
?- \+ (employee(X),salary(X,300)).

X = _G157
```

```
Yes
```

А этот запрос "не существует ни одного работника с окладом 300". Таких работников, действительно, нет. Переменная X осталась свободной (не получила никакого значения), поэтому в качестве значения выводится нечто маловразумительное.

Как видно, отрицание позволяет формулировать общие (общеотрицательные в терминологии классической логики) суждения. Выясним, кто не является начальником и кто не имеет начальников.

```
?- employee(X), \+ boss(_,X).

X = diana ;
X = frederic ;
X = herbert ;
X = isabella ;
X = john ;
No

?- employee(X), \+ boss(X,_).


X = anthony ;
No
```

Найдем самого высокооплачиваемого работника.

```
?- salary(A,Smax), \+ (salary(_,S), S>Smax).

A = anthony
Smax = 500
S = _G161

Yes
```

Интересное применение отрицания - конструкция (`\+ (Q, \+ P)`), которая проверяет, что все решения Q будут решениями для P. Например

```
?- \+ (employee(X), \+ salary(X,_)).

X = _G157
Yes
```

Это буквально означает "не существует работника, не имеющего оклада" или "все работники имеют оклады". В SWI-Prolog есть встроенный предикат `forall`, имеющий тот же смысл.

```
?- forall(employee(X), salary(X,_)).
```

```
X = _G157
Yes
```

Кроме фактов - безусловно истинных предложений, в программы можно включать и условные предложения - *правила*, которые определяют новые предикаты в терминах уже существующих. Каждое правило состоит из двух частей, разделенных символом ":-". Левая часть правила - *заголовок* описывает предикат, а правая - *тело* определяет условия, при которых он истинен. Телом может быть любой запрос, который и выполняется при вызове предиката, определенного в заголовке. Например, можно оформить в виде правил, вопросы, которые мы задавали.

```
all_paid :- forall(employee(X), salary(X,_)).


costly(A) :- boss(A,B), salary(A,SA), salary(B,SB), SA >
SB.


noboss(A) :- employee(A), \+ boss(_,A).


boss(B) :-  boss(_,B).
```

Дополнив программу новыми правилами, сможем задавать новые вопросы. При этом не интересующие нас переменные оказываются спрятанными внутри правил и не выводятся.

```
?- all_paid.


Yes



?- costly(A).


A = john ;
No



?- noboss(A).


A = diana ;
A = frederic ;
A = herbert ;
A = isabella ;
A = john ;
No
```

```
?- boss(charles).

Yes

?- boss(B).

B = anthony ;
B = anthony ;
B = barbara ;
B = barbara ;
B = charles ;
B = charles ;
B = charles ;
B = gregory ;
B = gregory ;
No
```

Обратите внимание, что в последнем запросе каждый ответ выдается несколько раз, ровно столько, сколько он встречается в программе. Нам еще придется столкнуться с этой особенностью - Пролог выдает ответ столько раз, сколькими способами он может его найти.

3.6 РАЗРАБОТКА ИНТЕЛЛЕКТУАЛЬНЫХ СЕРВИСОВ

Байесовская классификация

Альтернативные названия: байесовское моделирование, байесовская статистика, метод байесовских сетей. Изначально байесовская классификация использовалась для формализации знаний экспертов в экспертных системах, сейчас байесовская классификация также применяется в качестве одного из методов Data Mining.

Так называемая наивная классификация или наивно-байесовский подход (naive-bayes approach) является наиболее простым вариантом метода, использующего байесовские сети. При этом подходе решаются задачи классификации, результатом работы метода являются так называемые "прозрачные" модели.

"Наивная" классификация - достаточно прозрачный и понятный метод классификации. "Наивной" она называется потому, что исходит из предположения о взаимной независимости признаков.

Свойства наивной классификации:

Использование всех переменных и определение всех зависимостей между ними.

Наличие двух предположений относительно переменных:

все переменные являются одинаково важными;

все переменные являются статистически независимыми, т.е. значение одной переменной ничего не говорит о значении другой.

Большинство других методов классификации предполагают, что перед началом классификации вероятность того, что объект принадлежит тому или иному классу, одинакова; но это не всегда верно.

Допустим, известно, что определенный процент данных принадлежит конкретному классу. Возникает вопрос, можем ли мы использовать эту информацию при построении модели классификации? Существует множество реальных примеров использования этих априорных знаний, помогающих классифицировать объекты. Типичный пример из медицинской практики. Если доктор отправляет результаты анализов пациента на дополнительное исследование, он относит пациента к какому-то определенному классу. Каким образом можно применить эту информацию? Мы можем использовать ее в качестве дополнительных данных при построении классификационной модели.

Отмечают такие достоинства байесовских сетей как метода Data Mining:

в модели определяются зависимости между всеми переменными, это позволяет легко обрабатывать ситуации, в которых значения некоторых переменных неизвестны;

байесовские сети достаточно просто интерпретируются и позволяют на этапе прогностического моделирования легко проводить анализ по сценарию "что, если";

байесовский метод позволяет естественным образом совмещать закономерности, выведенные из данных, и, например, экспертные знания, полученные в явном виде;

использование байесовских сетей позволяет избежать проблемы переучивания (overfitting), то есть избыточного усложнения модели, что является слабой стороной многих методов (например, деревьев решений и нейронных сетей).

Наивно-байесовский подход имеет следующие недостатки:

перемножать условные вероятности корректно только тогда, когда все входные переменные действительно статистически независимы; хотя часто данный метод показывает достаточно хорошие результаты при несоблюдении условия статистической независимости, но теоретически такая ситуация должна обрабатываться более сложными методами, основанными на обучении байесовских сетей;

невозможна непосредственная обработка непрерывных переменных - требуется их преобразование к интервальной шкале, чтобы атрибуты были дискретными; однако такие преобразования иногда могут приводить к потере значимых закономерностей;

на результат классификации в наивно-байесовском подходе влияют только индивидуальные значения входных переменных, комбинированное влияние пар или троек значений разных атрибутов здесь не учитывается. Это могло бы улучшить качество классификационной модели с точки зрения ее прогнозирующей точности, однако, увеличило бы количество проверяемых вариантов.

Байесовская классификация нашла широкое применение на практике.

Байесовская фильтрация по словам

Не так давно байесовская классификация была предложена для персональной фильтрации спама. Первый фильтр был разработан Полем Грахемом (Paul Graham). Для работы алгоритма требуется выполнение двух требований.

Первое требование - необходимо, чтобы у классифицируемого объекта присутствовало достаточное количество признаков. Этому идеально удовлетворяют все слова писем пользователя, за исключением совсем коротких и очень редко встречающихся.

Второе требование - постоянное переобучение и пополнение набора "спам - не спам". Такие условия очень хорошо работают в локальных почтовых клиентах, так как поток "не спама" у конечного клиента достаточно постоянен, а если изменяется, то не быстро.

Однако для всех клиентов сервера точно определить поток "не спама" довольно сложно, поскольку одно и то же письмо, являющееся для одного клиента спамом, для другого спамом не является. Словарь получается слишком большим, не существует четкого разделения на спам и "не спам", в результате качество классификации, в данном случае решение задачи фильтрации писем, значительно снижается.

Методы классификации и прогнозирования. Деревья решений

Метод деревьев решений (decision trees) является одним из наиболее популярных методов решения задач классификации и прогнозирования. Иногда этот метод Data Mining также называют деревьями решающих правил, деревьями классификации и регрессии. Как видно из последнего названия, при помощи данного метода решаются задачи классификации и прогнозирования.

Если зависимая, т.е. целевая переменная принимает дискретные значения, при помощи метода дерева решений решается задача классификации.

Если же зависимая переменная принимает непрерывные значения, то дерево решений устанавливает зависимость этой переменной от независимых переменных, т.е. решает задачу численного прогнозирования.

Впервые деревья решений были предложены Ховилендом и Хантом (Hoveland, Hunt) в конце 50-х годов прошлого века. Самая ранняя и известная работа Ханта и др., в которой излагается суть деревьев решений - "Эксперименты в индукции" ("Experiments inInduction") - была опубликована в 1966 году.

В наиболее простом виде дерево решений - это способ представления правил в иерархической, последовательной структуре. Основа такой структуры - ответы "Да" или "Нет" на ряд вопросов.

На рисунке 8.1 приведен пример дерева решений, задача которого - ответить на вопрос: "Играть ли в гольф?" Чтобы решить задачу, т.е. принять решение, играть ли в гольф, следует отнести текущую ситуацию к одному из известных классов (в данном случае - "играть" или "не играть"). Для этого требуется ответить на ряд вопросов, которые находятся в узлах этого дерева, начиная с его корня.

Первый узел нашего дерева "Солнечно?" является узлом проверки, т.е. условием. При положительном ответе на вопрос осуществляется переход к левой части дерева, называемой левой ветвью, при отрицательном - к правой части дерева. Таким образом, внутренний узел дерева является узлом проверки определенного условия. Далее идет следующий вопрос и т.д., пока не будет достигнут конечный узел дерева, являющийся узлом решения. Для нашего дерева существует два типа конечного узла: "играть" и "не играть" в гольф.

В результате прохождения от корня дерева (иногда называемого корневой вершиной) до его вершины решается задача классификации, т.е. выбирается один из классов - "играть" и "не играть" в гольф.

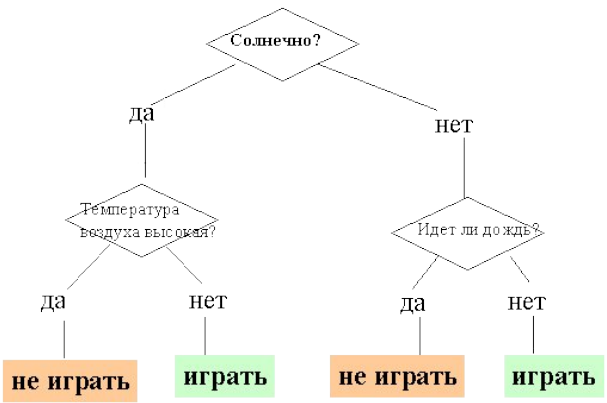

Рисунок 6.1 Дерево решений "Играть ли в гольф?"

Целью построения дерева решения в нашем случае является определение значения категориальной зависимой переменной.

Итак, для нашей задачи основными элементами дерева решений являются:

Корень дерева: "Солнечно?"

Внутренний узел дерева или узел проверки: "Температура воздуха высокая?", "Идет ли дождь?"

Лист, конечный узел дерева, узел решения или вершина: "Играть", "Не играть"

Ветвь дерева (случаи ответа): "Да", "Нет".

В рассмотренном примере решается задача бинарной классификации, т.е. создается дихотомическая классификационная модель. Пример демонстрирует работу так называемых бинарных деревьев.

В узлах бинарных деревьев ветвление может вестись только в двух направлениях, т.е. существует возможность только двух ответов на поставленный вопрос ("да" и "нет").

Построение классификационной модели

Бинарные деревья являются самым простым, частным случаем деревьев решений. В остальных случаях, ответов и, соответственно, ветвей дерева, выходящих из его внутреннего узла, может быть больше двух.

Рассмотрим более сложный пример. База данных, на основе которой должно осуществляться прогнозирование, содержит следующие ретроспективные данные о клиентах банка, являющиеся ее атрибутами: возраст, наличие недвижимости, образование, среднемесячный доход, вернул ли клиент вовремя кредит. Задача состоит в том, чтобы на основании перечисленных выше данных (кроме последнего атрибута) определить, стоит ли выдавать кредит новому клиенту.

Такая задача решается в два этапа: построение классификационной модели и ее использование.

На этапе построения модели, собственно, и строится дерево классификации или создается набор неких правил. На этапе использования модели построенное дерево, или путь от его корня к одной из вершин, являющийся набором правил для конкретного клиента, используется для ответа на поставленный вопрос "Выдавать ли кредит?"

Правилом является логическая конструкция, представленная в виде "если…: то…:".

На рисунке 9.2 приведен пример дерева классификации, с помощью которого решается задача "Выдавать ли кредит клиенту?". Она является типичной задачей классификации, и при помощи деревьев решений получают достаточно хорошие варианты ее решения.

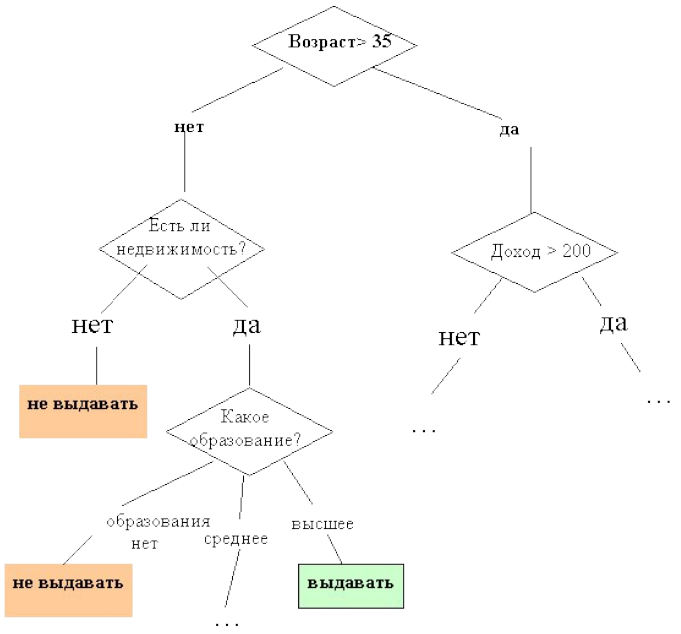

Рисунок 6.2 Дерево решений "Выдавать ли кредит?"

Как мы видим, внутренние узлы дерева (возраст, наличие недвижимости, доход и образование) являются атрибутами описанной выше базы данных. Эти атрибуты называют прогнозирующими, или атрибутами расщепления (splitting attribute). Конечные узлы дерева, или листы, именуются метками класса, являющимися значениями зависимой категориальной переменной "выдавать" или "не выдавать" кредит.

Каждая ветвь дерева, идущая от внутреннего узла, отмечена предикатом расщепления. Последний может относиться лишь к одному атрибуту расщепления данного узла. Характерная особенность предикатов расщепления: каждая запись использует уникальный путьот корня дерева только к одному узлу-решению. Объединенная информация об атрибутах расщепления и предикатах расщепления в узле называется критерием расщепления (splitting criterion).

На рисунке 6.1 изображено одно из возможных деревьев решений для рассматриваемой базы данных. Например, критерий расщепления "Какое образование?", мог бы иметь два предиката расщепления и выглядеть иначе: образование "высшее" и "не высшее". Тогда дерево решений имело бы другой вид.

Таким образом, для данной задачи (как и для любой другой) может быть построено множество деревьев решений различного качества, с различной прогнозирующей точностью.

Качество построенного дерева решения весьма зависит от правильного выбора критерия расщепления. Над разработкой и усовершенствованием критериев работают многие исследователи.

Метод деревьев решений часто называют "наивным" подходом. Но благодаря целому ряду преимуществ, данный метод является одним из наиболее популярных для решения задач классификации.

Преимущества деревьев решений

Интуитивность деревьев решений. Классификационная модель, представленная в виде дерева решений, является интуитивной и упрощает понимание решаемой задачи. Результат работы алгоритмов конструирования деревьев решений, в отличие, например, от нейронных сетей, представляющих собой "черные ящики", легко интерпретируется пользователем. Это свойство деревьев решений не только важно при отнесении к определенному классу нового объекта, но и полезно при интерпретации модели классификации в целом. Дерево решений позволяет понять и объяснить, почему конкретный объект относится к тому или иному классу.

Деревья решений дают возможность извлекать правила из базы данных на естественном языке. Пример правила: Если Возраст > 35 и Доход > 200, то выдать кредит.

Деревья решений позволяют создавать классификационные модели в тех областях, где аналитику достаточно сложно формализовать знания.

Алгоритм конструирования дерева решений не требует от пользователя выбора входных атрибутов (независимых переменных). На вход алгоритма можно подавать все существующие атрибуты, алгоритм сам выберет наиболее значимые среди них, и только они будут использованы для построения дерева. В сравнении, например, с нейронными сетями, это значительно облегчает пользователю работу, поскольку в нейронных сетях выбор количества входных атрибутов существенно влияет на время обучения.

Точность моделей, созданных при помощи деревьев решений, сопоставима с другими методами построения классификационных моделей (статистические методы, нейронные сети).

Разработан ряд масштабируемых алгоритмов, которые могут быть использованы для построения деревьев решения на сверхбольших базах данных; масштабируемость здесь означает, что с ростом числа примеров или записей базы данных время, затрачиваемое на обучение, т.е. построение деревьев решений, растет линейно. Примеры таких алгоритмов: SLIQ, SPRINT.

Быстрый процесс обучения. На построение классификационных моделей при помощи алгоритмов конструирования деревьев решений требуется значительно меньше времени, чем, например, на обучение нейронных сетей.

Большинство алгоритмов конструирования деревьев решений имеют возможность специальной обработки пропущенных значений.

Многие классические статистические методы, при помощи которых решаются задачи классификации, могут работать только с числовыми данными, в то время как деревья решений работают и с числовыми, и с категориальными типами данных.

Многие статистические методы являются параметрическими, и пользователь должен заранее владеть определенной информацией, например, знать вид модели, иметь гипотезу о виде зависимости между переменными, предполагать, какой вид распределения имеют данные. Деревья решений, в отличие от таких методов, строят непараметрические модели. Таким образом, деревья решений способны решать такие задачи Data Mining, в которых отсутствует априорная информация о виде зависимости между исследуемыми данными.

Процесс конструирования дерева решений

Напомним, что рассматриваемая нами задача классификации относится к стратегии обучения с учителем, иногда называемого индуктивным обучением. В этих случаях все объекты тренировочного набора данных заранее отнесены к одному из предопределенных классов.

Алгоритмы конструирования деревьев решений состоят из этапов "построение" или " создание " дерева (tree building) и "сокращение " дерева (tree pruning). В ходе создания дерева решаются вопросы выбора критерия расщепления и остановки обучения (если это предусмотрено алгоритмом). В ходе этапа сокращения дерева решается вопрос отсечения некоторых его ветвей.

Рассмотрим эти вопросы подробней.

Критерий расщепления

Процесс создания дерева происходит сверху вниз, т.е. является нисходящим. В ходе процесса алгоритм должен найти такойкритерий расщепления, иногда также называемый критерием разбиения, чтобы разбить множество на подмножества, которые бы ассоциировались с данным узлом проверки. Каждый узел проверки должен быть помечен определенным атрибутом. Существуетправило выбора атрибута: он должен разбивать исходное множество данных таким образом, чтобы объекты подмножеств, получаемых в результате этого разбиения, являлись представителями одного класса или же были максимально приближены к такому разбиению. Последняя фраза означает, что количество объектов из других классов, так называемых "примесей", в каждом классе должно стремиться к минимуму.

Существуют различные критерии расщепления. Наиболее известные - мера энтропии и индекс Gini.

В некоторых методах для выбора атрибута расщепления используется так называемая мера информативности подпространств атрибутов, которая основывается на энтропийном подходе и известна под названием "мера информационного выигрыша" (information gain measure) или мера энтропии.

4. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

4.1. Перечень основной и дополнительной литературы, необходимой для освоения дисциплины

4.1.1. Перечень основной литературы

1. Цуканова Н.М, Дмитриева Т.А. Логическое программирование на языке Visual Prolog. Серия: Специальность. Для высших учебных заведений – М.: Горячая линия-Телеком, 2016.

2. Макконнелл, С. Совершенный код. Мастер-класс. / Стив Макконнелл. - М.: Русская Редакция, 2016.

4.1.2. Перечень дополнительной литературы

1. Джозеф Джарратано, Гари Райли. Экспертные системы. Принципы разработки и программирование, 4-е издание. – М.: Вильямс, 2017.

2. Чезарини Ф., Томпсон С. Программирование в Erlang. М.: ДМК Пресс, 2016.

4.2. Перечень учебно-методического обеспечения самостоятельной работы обучающихся по дисциплине

1. Методические указания по выполнению лабораторных работ по дисциплине «Введение в функциональное программирование».

2. Методические рекомендации для студентов по организации самостоятельной работы по дисциплине «Введение в функциональное программирование».

4.3. Перечень ресурсов информационно-телекоммуникационной сети Интернет, необходимых для освоения дисциплины

1. Национальный Открытый Университет. Интуит. http://www.intuit.ru.
2. Федеральный портал «Российское образование. http://www.edu.ru.
3. Российская государственная библиотека. http://www.rsl.ru.

x: *x*: