

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

**Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный
лесотехнический университет» им. С.М. Кирова**

Кафедра информационных систем и технологий

С.П.Хабаров, кандидат технических наук, доцент

**ИНТЕЛЛЕКТУАЛЬНЫЕ
ИНФОРМАЦИОННЫЕ СИСТЕМЫ**

**PROLOG – ЯЗЫК РАЗРАБОТКИ
ИНТЕЛЛЕКТУАЛЬНЫХ И ЭКСПЕРТНЫХ СИСТЕМ**

Учебное пособие
для бакалавров и магистров направлений подготовки
230400 «Информационные системы и технологии»
и 230200 «Информационные системы»

Санкт-Петербург
2013

Рассмотрено и рекомендовано к изданию
Учебно-методической комиссией лесохозяйственного факультета
Санкт-Петербургского государственного лесотехнического университета
12 октября 2013г.

Отв. редактор
Кандидат технических наук, доцент **С.П. Хабаров**

Рецензенты:
кафедра Информационных систем и вычислительной техники
Национального минерально-сырьевого университета «Горный»
(доктор технических наук, профессор И.В. Иванова),
доктор технических наук, профессор, **А.Ю. Тропченко** (НУИИТМО)

УДК 004.89(075.8)

Хабаров С.П.

Интеллектуальные информационные системы. PROLOG- язык
разработки интеллектуальных и экспертных систем: учебное пособие /
С.П.Хабаров.- СПб. СПбГЛТУ, 2013.- 138 с.

ISBN

Представлено кафедрой информационных систем и технологий.

В пособии дана характеристика современного этапа развития языка Prolog и его возможностей для разработки интеллектуальных систем. Приведены общие сведения о конструкции языка и описан процесс разработки программ на этом языке. Рассмотрены наиболее распространенные среды программирования: PIE, Turbo-Prolog, SWI-Prolog и Visual-Prolog. Большое число примеров и заданий позволяет изучить основные методы работы в каждой из этих сред, а также познакомиться с особенностями их использования для реализации конкретных проектов.

Учебное пособие предназначено для бакалавров и магистров направлений подготовки 230400 «Информационные системы и технологии» и 230200 «Информационные системы». Оно будет полезно аспирантам, преподавателям вуза и специалистам, применяющим методы искусственного интеллекта в своей профессиональной деятельности.

Табл. 7. Ил. 81. Библиогр. 12 назв.

Темплан 2013 г. Изд. №
ISBN

©СПбГЛТУ, 2013

СОДЕРЖАНИЕ

1. История развития и современное состояние языка Prolog	4
1.1. Тенденции в истории развития языка Prolog	4
1.2. Области применения языка Prolog	7
1.3. Современные реализации языка Prolog	10
2. Основы разработки Prolog-программ	18
2.1. Общие сведения о языке Prolog	18
2.2. Понятие факта, правила, запроса и процедуры	19
2.3. Механизм сопоставления и поиска с возвратом	21
2.4. Основные элементы языка Prolog	23
3. Среда разработки программ PIE и система SWI-Prolog	27
3.1. Основы работы в консоли Prolog	29
3.2. Основы работы с программными файлами	33
3.3. Справка и помощь в среде SWI-Prolog	40
3.4. Основы трассировки и отладки в среде PIE и SWI-Prolog	42
3.5. Графические интерфейсы на базе SWI-Prolog/XPCE	46
3.6. Загрузка и запуск программ SWI-Prolog'a	54
4. Среда разработки Turbo-Prolog	64
4.1. Структура программы на Turbo Prolog'e	65
4.2. Оболочка системы Turbo Prolog	71
4.3. Отладка и трассировка программ	74
4.4. Работа с простейшими программами в Turbo Prolog'e	75
4.5. Работа с графикой в Turbo Prolog'e	84
5. Среда разработки Visual Prolog	87
5.1. Интегрированная среда разработки	87
5.2. Разработка консольного проекта	88
5.3. Разработка GUI проекта на Visual Prolog'e	94
6. Prolog — язык логического программирования	105
6.1. Системы основанные на знаниях	105
6.2. Логические модели и логическое программирование	106
6.3. Простейшие конструкции языка предикатов	107
6.4. Предикатные формулы	108
6.5. Определение правильно построенной формулы	110
6.6. Логический вывод	110
6.7. Решение задач и извлечение ответа	116
Приложение	118
Приложение 1. Запуск Turbo-Prolog в ОС Windows 7	118
Приложение 2. Служебные предикаты Турбо-Пролога	124
Приложение 3. Пример программы построения дерева синтаксического анализа	132
Библиографический список	137

1. ИСТОРИЯ РАЗВИТИЯ И СОВРЕМЕННОЕ СОСТОЯНИЕ ЯЗЫКА PROLOG

1.1. Тенденции в истории развития языка Prolog

Любой язык программирования всегда ориентирован на определенный круг задач, при решении которых он наиболее эффективен. Для языка Prolog типичными являются проекты, связанные с разработкой систем искусственного интеллекта — это различные экспертные системы, системы планирования, программы-переводчики, интеллектуальные игры и т.п. Prolog обладает достаточно мощными средствами, позволяющими извлекать информацию из баз данных и знаний. При этом его методы поиска принципиально отличаются от "традиционных". Кроме этого, Prolog часто используют в задачах, связанных с манипулированием на естественном языке.

Интерес к Prolog поднимался и затихал несколько раз, энтузиазм сменялся жёстким неприятием. Наиболее высокий интерес к языку Prolog, как к языку будущего, возник в 1980-х годах во время разработки японской национальной программы «компьютеры пятого поколения». В рамках этого проекта разработчики надеялись, что с помощью Prolog можно будет сформулировать новые принципы, которые приведут к созданию компьютеров более высокого уровня интеллекта.

Эта надежда базировалась на том, что логическое программирование использует не последовательность абстракций и преобразований, которая отталкивается от машинной архитектуры фон Неймана и присущего ей набора операций, а на основе абстрактной модели, которая никак не связана с каким-либо типом машинной модели. Из этого подхода следовало, что не человека надо обучать мышлению в терминах операций компьютера, а компьютер должен выполнять инструкции, свойственные человеку. И на этом историческом этапе некоторые ученые и инженеры считали подобный путь простым и эффективным.

Prolog — язык программирования, который основан не на алгоритме, а на логике предикатов. Если программа на алгоритмическом (процедурном) языке является последовательностью инструкций, выполняющихся в заданном порядке, то Prolog, будучи декларативным языком, содержит только описание задачи, а Prolog-машина выполняет поиск решения, руковод-

ствуясь только этим описанием и используя механизм поиска с возвратом и унификацию.

Разработка языка Prolog началась в 1970 г. Аланом Кулмероэ (Alain Colmerauer) и Филиппом Русселом (Philippe Roussel) в университете города Марсель, Франция. Они хотели создать язык, который мог бы делать логические заключения на основе заданного текста. Их работа была частично мотивирована желанием примирить использование логики в качестве декларативного языка представления знаний с процессуальным подходом к представлению знаний, который был популярен в Северной Америке в конце 1960-х и начале 1970-х годов.

Название Prolog было выбрано Филиппом Русселом как аббревиатура от французского PROgrammation en LOGique (PROLOG) и первая реализация этого языка с использованием компилятора Николауса Вирта "Algol-W" была закончена в 1972 году. Основы современного языка были заложены в 1973 году. Prolog постепенно распространялся среди тех, кто занимался логическим программированием, в основном благодаря личным контактам, а не через коммерциализацию продукта.

Только в 1977 году Д. Уоррен и Ф. Перейра в университете Эдинбурга создают очень эффективный компилятор языка Prolog для ЭВМ DEC-10 и переводят методы логического программирования в практическую плоскость. Интересно, что компилятор был написан на самом Prolog. Эта реализация Prolog, известная как "эдинбургская версия", фактически стала первым и единственным стандартом языка. Как правило, если современная Prolog-система и не поддерживает эдинбургский Prolog, то в ее состав входит подсистема, переводящая Prolog-программу в "эдинбургский" вид.

Позднее в 1980 году К. Кларк и Ф. Маккейб в Великобритании разработали версию Prologa для персональных ЭВМ. А в 1996 году международной организацией по стандартизации (ISO - International Organization for Standardization) была создана по языку программирования Prolog рабочая группа ISO/IEC JTC1/SC22/WG17 и принят стандарт ISO/IEC 13211-1 (Part 1: General core). Этот стандарт имеет две коррекции в 2007 и 2012 годах. Кроме этого в 2000 году был принят дополнительный стандарт ISO/IEC 13211-2 (Part 2: Modules), который стандартизует использование в Prolog модулей.

Как видно из приведенных выше стандартов, слухи о безвременной кончине Prolog не являются достоверными. Он и сейчас остается наиболее популярным языком искусственного интеллекта в Японии и Европе. В США, традиционно, более распространен другой язык искусственного интеллекта — язык функционального программирования Лисп.

В развитии языка Prolog наблюдаются очень интересные тенденции. Этот язык быстро приобрел популярность в Европе как инструмент прак-

тического программирования. В Японии вокруг языка Prolog были сосредоточены все разработки компьютеров пятого поколения. С другой стороны, в США этот язык в целом был принят с небольшим опозданием в связи с некоторыми историческими причинами [1].

Одна из них состояла в том, что США вначале познакомились с языком Microplanner, который был близок к идее логического программирования, но неэффективно реализован. Определенная доля низкой популярности Prolog в этой стране объясняется также реакцией на существовавшую вначале "ортодоксальную школу" логического программирования, представители которой настаивали на использовании чистой логики и требовали, чтобы логический подход не был "запятнан" практическими средствами, не относящимися к логике.

В прошлом это привело к распространению неверных взглядов на язык Prolog. Некоторые считали, что на этом языке можно программировать только рассуждения с выводом от целей к фактам. Но истина в том, что Prolog является универсальным языком программирования и на нем можно реализовать любой алгоритм. Позиция "ортодоксальной школы" была преодолена практиками, которые использовали более прагматический подход, объединив декларативный подход с традиционным, процедурным. Одной из самых мощных реализаций Prolog в настоящее время является система Visual Prolog, представляя собой полнофункциональную среду программирования. При этом он имеет богатую историю и является развитием и наследником таких систем, как Turbo Prolog и PDC Prolog.

- Turbo Prolog 1.0 (1986 г.) — Разработан датской компанией Prolog Development Center (PDC) в содружестве с фирмой Borland International. Система создавалась с серьезными отступлениями от неофициального стандарта языка, самым существенным из которых было введение строгой типизации данных, но это позволило значительно ускорить трансляцию и выполнение программ. Новый компилятор был по достоинству оценен программистами-практиками, хотя и критиковался в академических кругах.

- Turbo Prolog 2.0 (1988 г.) — Более мощная версия, которая содержала усовершенствованную интегрированную среду разработки программ, быстрый компилятор и средства низкоуровневого программирования. Кроме того, она предоставляла возможность работы с собственными внешними БД, dBase III и Reflex, интегрированным пакетом Lotus 1-2-3, графическим пакетом Paint Brush и другими приложениями. Фирма Borland распространяла эту версию до 1990 г., а затем компания PDC приобрела монопольное право на использование исходных текстов компилятора и дальнейшее продвижение системы программирования на рынок.

- PDC Prolog 3.31 (1992 г.) — Это был эффективный универсальный инструмент профессиональных программистов, который вскоре стал одним из наиболее широко используемых и мог работать в среде MS DOS, OS/2, UNIX, XENIX, PharLap DOS Extender и MS Windows. Эта версия была хорошо совместима с традиционными языками программирования, в первую очередь с языком С. В ней были расширены возможности создания приложений с интерфейсом GUI, принятым в MS Windows и OS/2

- Visual Prolog 4.0 (1996 г.) — В этой работе участвовали российские программисты под руководством Виктора Юхтенко, который позже стал техническим директором компании «Пролог-Софт», представляющей интересы PDC в России. В эту версию входят различные элементы: прежде всего, интерактивная среда визуальной разработки (VDE — Visual Develop Environment), которая включает текстовый и различные графические редакторы, инструментальные средства генерации кода, конструирующие управляющую логику (Experts), а также являющийся расширением языка интерфейс визуального программирования (VPI — Visual Programming Interface), Пролог-компилятор, набор различных подключаемых файлов и библиотек, редактор связей, файлы, содержащие примеры и помощь. Visual Prolog 4.0 поддерживается различными ОС, в том числе MS-DOS PharLap-Extended DOS, всеми версиями Windows, 16 и 32-битовыми целевыми платформами OS/2, а также некоторыми другими системами, требующими графического пользовательского интерфейса.

- Visual Prolog 7.4 (2013 г.) — Последняя версия, представленная на сайте PDC на момент написания данной книги.

1.2. Области применения языка Prolog

Базовым принципом языка является равнозначность представления программы и данных (декларативность), отчего утверждения языка одновременно являются и записями, подобными записям в базе данных, и правилами, несущими в себе способы их обработки. Сочетание этих качеств приводит к тому, что по мере работы Prolog-системы знания накапливаются. Накапливаются как данные, так и правила. Поэтому Prolog-системы считают естественной средой для накопления базы знаний. База знаний — важный компонент интеллектуальной системы. Они предназначены для поиска способов решения проблем из некоторой предметной области, основываясь на записях базы знаний и на пользовательском описании ситуации.

Простые базы знаний могут использоваться для создания экспертных систем хранения данных в организации: документации, руководств, статей технического обеспечения. Главная цель создания таких баз — помочь ме-

нее опытным людям найти уже существующее описание способа решения какой-либо проблемы

Говоря об областях применения Prolog следует сразу оговориться, что он слабо приспособлен для решения задач, связанных с обработкой графики, вычислениями или численными методами. Вместе с тем он с успехом может использоваться в компьютерной алгебре, которая в отличие от численных методов, занимается реализацией аналитических методов решения математических задач на компьютере и предполагает, что исходные данные, как и результаты решения, сформулированы в аналитическом виде. В качестве основных областей применения Prolog можно отметить следующие направления:

- разработка быстрых прототипов прикладных программ;
- управление производственными процессами;
- создание динамических реляционных баз данных;
- перевод с одного языка на другой;
- создание естественно-языковых интерфейсов;
- реализация экспертных систем и оболочек экспертных систем;
- создание пакетов символьных вычислений;
- доказательства теорем и интеллектуальные системы, в которых возможности языка Prolog по обеспечению дедуктивного вывода применяются для проверки различных теорий.

Prolog нашел применение и в ряде других областей, например, при решении задач составления сложных расписаний. Он используется в различных системах, но обычно не в качестве основного языка, а в качестве языка для разработки некоторой части системы. Достаточно часто Prolog используют для написания функций взаимодействия с базами данных.

Используют его и в сложных поисковых системах, которые выполняют не только поиск, но и играют роль некоторой "отвечающей системы" – программного комплекса, который умеет извлекать информацию из большой выборки текстовых файлов и баз данных, а затем вести диалог с пользователем, отвечая, в обычном понимании этого слова, на его вопросы.

Также Prolog используют при написании новых специфичных языков программирования. Например, функциональный язык Erland построен на основе Prolog. По сути, Erland является усовершенствованием Prolog для некоторых специфических целей, связанных с задачами реального времени.

В данное время Prolog, несмотря на неоднократные пессимистические прогнозы, продолжает развиваться в разных странах, включая в себя новые технологии и концепции. К ним относятся и парадигмы императивного программирования, при котором процесс вычисления описывают в виде инструкций, изменяющих состояние программы.

Одно из направлений развития языка, в том числе и в России, реализует концепцию интеллектуальных агентов. Под этим термином понимаются разумные сущности, наблюдающие за окружающей средой и действующие в ней. При этом их поведение рационально в том смысле, что они способны к пониманию, а их действия направлены на достижение какой-либо цели. Такой агент может быть как роботом, так и встроенной программой. Об интеллектуальности агента можно говорить, если он взаимодействует с окружающей средой примерно так же, как действовал бы человек.

В качестве примера, иллюстрирующего области использования Prolog рассмотрим ряд проектов, реализованных датской компанией PDC, домашняя страница которой приведена на рис.1.1. Вместе с разработкой собственно системы программирования Visual Prolog, компания PDC на основе Visual Prolog разработала и внедрила ряд интеллектуальных продуктов и решений. Основными среди них являются.:



Рис. 1.1. Домашняя страница компании PDC (www.pdc.dk).

- PDC SCORE и другие – это ИТ-решения в области авиации, которые предназначены для планирования и составления расписаний для бизнес-авиации, а также для бизнес-планирования работы авиакомпаний, аэропортов и наземного обслуживания самолетов. Около 20% мировых авиаперевозок координируется с использованием PDC SCORE. Более 40 международных авиакомпаний и 280 аэропортов используют эти решения.

- ARGOS – информационная система управления и принятия решений в кризисных ситуациях.
- PDC StaffPlan – это ИТ-решение для экономически эффективного кадрового планирования и управления ресурсами. Используется уже около 20 лет в крупных и средних организациях и компаниях в сфере здравоохранения, розничной торговли, аэропортах и промышленности.
- Dictus – это программа распознавания и синтеза речи, позволяющая превращать речь в печатный текст с 98% точностью, управлять компьютером с помощью голосовых команд, использовать голосовую передачу и прием SMS-сообщений и т.п.

В этих проектах используется разработанная в PDC технология, которая позволяет применять правила и методы искусственного интеллекта для оптимальной и эффективной поддержки принятия решений. Основу этой технологии составляет Visual Prolog. Сегодня, по мнению PDC, Visual Prolog представляет собой мощный и безопасный язык программирования сочетающий лучшие возможности логического, функционального и объектно-ориентированного программирования.

Если говорить о менее мощных реализациях Prolog, поддерживающих в основном эдинбургский стандарт ISO Prolog, то, вне всякого сомнения, программа на языке Prolog может стать "мозгом" различных чрезвычайно интересных приложений. Но при этом выполнение интерфейсных функций (ввод-вывод, преобразование форматов данных, диалог с пользователем и т.д.) целесообразно возложить на другие языки программирования.

В настоящее время подобная практика разработки прикладных программных комплексов на основе Prolog находит широкое распространение. Этому способствуют и современные реализации языка Prolog, которые хорошо сочетаются с Java, Delphi, C#, C++ и др. Это позволяет реализовать в разрабатываемых приложениях и процедурные, то есть алгоритмические функции, и методы решения декларативных, интеллектуальных задач.

1.3. Современные реализации языка Prolog

Принятие стандарта языка Prolog стало стимулом для создания многих независимых реализаций этого языка, как коммерческих, так и бесплатно предоставляемых широкому кругу пользователей. Постоянно обновляемый обзор новейших реализаций Prolog можно найти по адресу:

<http://dmoz.org/Computers/Programming/Languages/Prolog/Implementations/>

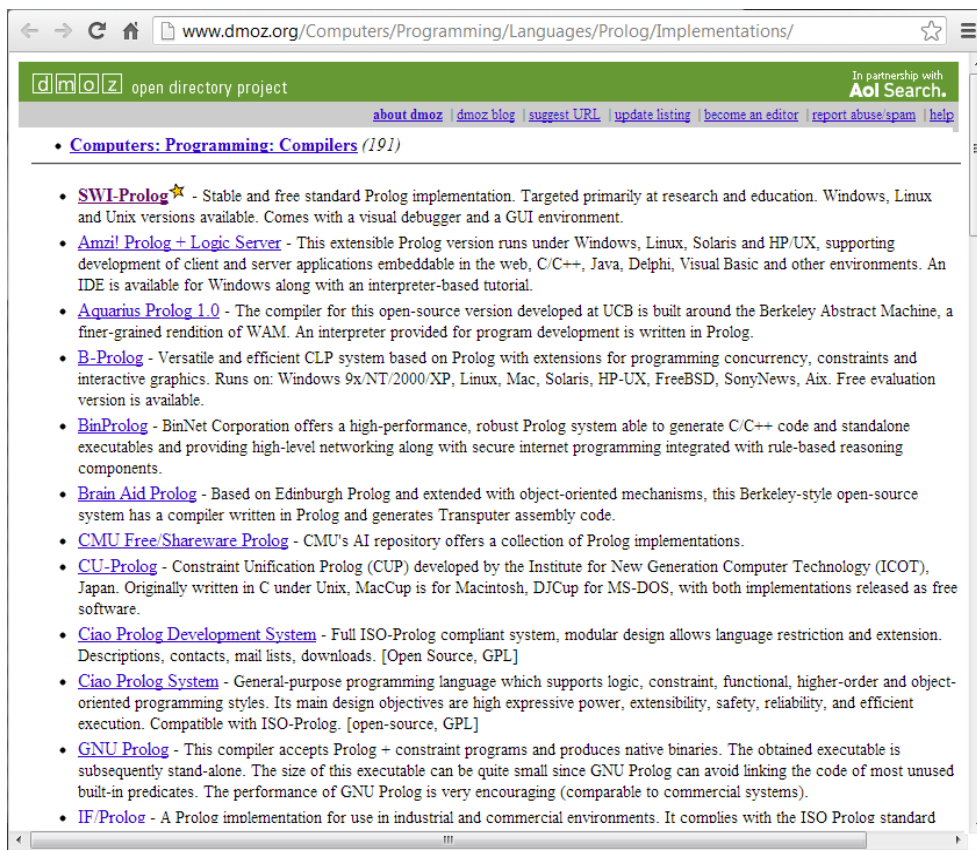


Рис. 1.2. Список современных дистрибутивов языка Prolog.

На момент написания книги этот список (рис. 1.2) насчитывал 26 реализаций. Среди них присутствуют как коммерческие версии, так и свободно распространяемые. Кроме того, для многих коммерческих дистрибутивов предусмотрены демонстрационные версии, с лицензией на ограниченный, но достаточно большой период времени, а также персональные выпуски (Personal Edition), которые предоставляются бесплатно для личного пользования.

Таблица 1.1

Сравнение основных характеристик реализаций Prolog

Платформа		Особенности								Инструмент			Механизм
Наименование	Операционная система	Графика	Компилятор	Unicode	ООП	Управление ОС	Автон. исполнение	Интерфейс к C	Интерфейс к Java	Интерпретатор	Отладчик	Профайлер	Синтаксис Prolog
BProlog	U, W, M		+		+	+	+	+	+	+	+	+	ISO
JProlog	JVM, A	+		+		через Java	+	через Java	+	+	+		ISO

Платформа		Особенности							Инструмент			Механизм
Ciao	U,W,M		+		+	+	+	+	+	+	+	ISO_R.
GNU Prolog	U,W,M		+			+	+	+	+	+		ISO
JLog	JVM	+	+					+	+			ISO
JScriptLog	Браузер								+			ISO
jTrolog	JVM			+				+	+	+		ISO
LPA-Prolog	W	+	+	+	+	+	+	+	+	+	+	EP_R
Open Prolog	Mac OS									+		
Poplog	L, U, W	+	+			+	+	+	+	+		EP_R
SICStus Prolog	U,L,W,M	+	+	+	+	+	+	+	+	+	+	ISO
SWI-Prolog	U,L,W,M	+	+	+		+	+	+	+	+	+	ISO, EP
tuProlog	JVM, A	+		+				+	+	+		ISO
Visual Prolog	W	+	+	+	+	+	+	+		+	+	
XSB Prolog	L,W,S,M		+			+	+	+	+	+	+	ISO
ПЕА-Prolog	L,W,S,M		+	+		+	+	+	+	+		EP, ISO

Условные обозначения: U – ОС Unix , W – ОС Windows, M -Mac OS X, L - ОС Linux, S – ОС Solaris, A – ОС Android, JVM - Java Virtual Machine, ISO – ISO Prolog, ISO_R - ISO Prolog с расширениями, EP - Edinburgh Prolog, EP_R - Edinburgh Prolog с расширениями

Достаточно полный обзор основных современных реализаций языка Prolog представлен в Интернете на портале Википедия по адресу: http://en.wikipedia.org/wiki/Comparison_of_Prolog_implementations. Контент этого ресурса позволяет сравнить основные характеристики различных реализаций Prolog (табл. 1.1) и познакомиться с возможностью поддержки веб-технологий (табл. 1.2) этими реализациями.

Таблица 1.2

Поддержка веб-технологий в реализациях Prolog

Наименование	Усл. компиляция	Сокеты	Многопоточность	HTTP клиент	HTTP сервер	HTML парсер	Адрес сайта в Интернете
B-Prolog							www.probp.com/
Ciao	+	+	+	+		+	www.ciao-lang.org/
GNU Prolog		+					www.gprolog.org
Jekejeke Prolog		+	+	+	+		www.jekejeke.ch

Наименование	Усл. компиляция	Сокеты	Многопоточность	HTTP клиент	HTTP сервер	HTML парсер	Адрес сайта в Интернете
LPA-Prolog		+		+	+	+	www.lpa.co.uk/win.htm
SICStus Prolog	+	+	+				www.sics.se/projects/sicstus-prolog-leading-prolog-technology
SWI-Prolog	+	+	+	+	+	+	www.swi-prolog.org/
Visual Prolog	+	+	+	+		+	www.visual-prolog.com/
XSB		+	+	+			xsb.sourceforge.net
YAP-Prolog	+	+	+				www.dcc.fc.up.pt/~vsc/Yap/

Наиболее мощные реализации Prolog позволяют использовать средства статистического анализа, а также средства логического программирования в ограничениях (Constraint Logic Programming - CLP). Средства CLP показали себя на практике как исключительно гибкий инструмент для решения задач принятия решений, составления расписаний, планирования материально-технического снабжения и решения комбинаторных задач искусственного интеллекта.

За последние 10-15 лет программирование в ограничениях прошло путь от научной идеи до мощной парадигмы программирования и считается в настоящее время одним из стратегических направлений информатики [2]. Наиболее популярными реализациями, включающими в свой состав CLP являются: B-Prolog, CHIP V5, Ciao Prolog, ECLiPSe, SICStus, GNU Prolog, YAP Prolog.

Так, например, в релиз B-Prolog v.7.8 добавлено моделирование задач линейного и смешанного целочисленного программирования, а также SAT-решатель. В любой реализации Prolog, поддерживающей CLP, отношения между переменными указываются в форме ограничений. Ограничения могут быть: вида, используемого в задачах удовлетворения ограничений (например, «А или В истинно»), вида, решаемых симплексным алгоритмом (например, « $x \leq 5$ ») и ряд других видов. Одним из ярких представителей таких систем является ECLiPSe (ECLiPSe Constraint Logic Programming System), интегрирующая различные расширения логической парадигмы программирования, в особенности логического программирования с ограничениями (рис.1.3). Ее ядро представляет собой эффективную реализацию Edinburgh Prolog. В основе лежит инкрементальный компиля-

тор исходных кодов в коды виртуальной машины. Система написана на Prolog и C.

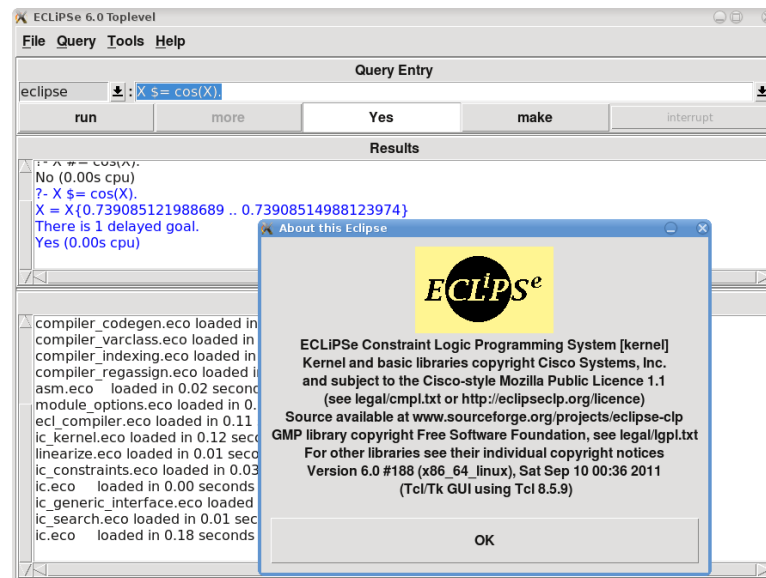


Рис. 1.3. TkECLiPSe - стандартная GUI-оболочка для ECLiPSe, версия 6.0.

Ряд реализаций Prolog включают в свой состав ряд дополнительных расширений. К числу которых относятся такие расширения, как:

- Запоминание результатов (tabling) - техника известная в функциональном программировании как мемоизация (memoization), которая освобождает пользователя от ручного хранения промежуточных результатов.
- Специально построенные базы данных (triplestore), для хранения и поиска троек «субъект - предикат - объект». Утверждение «Небо голубого цвета» будет представлено: «небо» (субъект), «имеет цвет» (предикат) и «голубой» (объект). В отличие от реляционных баз данных, triplestore оптимизирована для хранения и поиска троек, который выполняется с помощью своего языка запросов. В дополнение к запросам, тройки обычно можно импортировать / экспортировать с помощью Resource Description Framework (RDF) и других форматов.

Отдельно следует отметить наличие в настоящее время реализаций Prolog для мобильных устройств. К их числу относятся ProLog for iPhone 1.0, а также Prolog Mobile 9.5 от Meridian Project Systems, Inc. и tuProlog for Android (рис. 1.4), JIProlog для J2ME (MIDP 2.0, CLDC 1.0) для мобильных телефонов с поддержкой Java и ряд других.

JIProlog (Java Internet Prolog) — кроссплатформенный интерпретатор языка Prolog, который интегрирует языки Prolog и Java, позволяя вызывать предикаты Prolog из Java и методы Java из Prolog. JIProlog спроектирован так, что он может работать с любой версией Java 1.1 и выше.



Рис. 1.4. Скриншоты tuProlog for Android и Prolog Mobile 9.5 на iPad.

На сайте JIProlog есть режим demo (<http://www.jiprolog.com/demo.aspx>), в котором загружается Java-плагин консоли Prolog. Он позволяет вводить и исследовать небольшие Prolog-задачи прямо из веб-браузера (рис. 1.5).

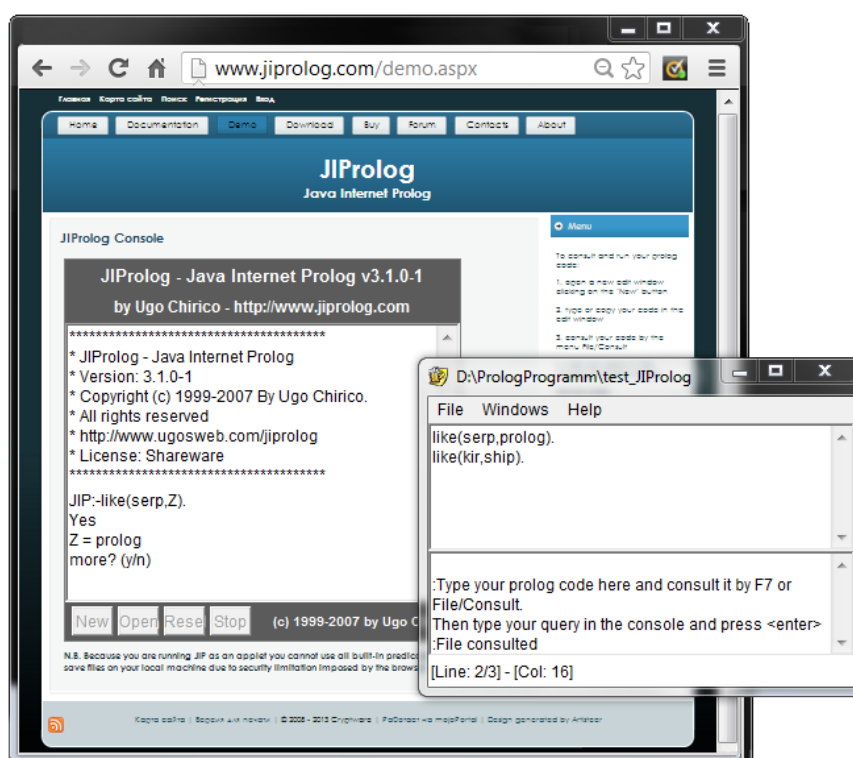


Рис. 1.5. Редактор и консоль JIProlog в веб-браузере.

В это же время разрабатываются множество коммерческих реализаций Prolog практически для всех типов компьютеров. Две коммерческие системы Quintus Prolog (<http://www.sics.se/quintus/>) и SICStus Prolog (<http://www.sics.se/sicstus/>) сейчас развиваются и поддерживаются SICS (Swedish Institute of Computer Science). Среды разработки с большим набором

ром инструментов и очень обширными библиотеками. Большинство разработчиков используют SICStus.

Одним из самых интересных продуктов является Visual Prolog. Это полнофункциональная среда программирования, которая предлагает полный набор средств, необходимых для разработки критически важных задач, коммерческого класса приложений. Несмотря на своё название это - не реализация Пролога, а совершенно иной язык со строгим контролем типов. Visual Prolog возник не на пустом месте, а является развитием и наследником таких языков программирования как PDC Prolog и Turbo Prolog.

Следует обратить внимание, что в списке на рис. 1.2, все реализации Prolog приведены в алфавитном порядке, и только одна из них выбивается из этого ряда: помещена в начало списка и отмечена звездочкой. Это SWI-Prolog – стабильная и бесплатная стандартная реализация Prolog, которая ориентирована в первую очередь на научные исследования и образование, но возможно и ее коммерческое использование. Для SWI-Prolog доступны версии под Windows, Linux и Unix, а также открытый исходный код.

Если на сайте SWI-Prolog обратиться к статистике загрузок этого продукта, то можно обнаружить, что в период с 2009 по 2012 годы количество скачиваний в среднем более 10 тыс. в месяц (рис. 1.6).

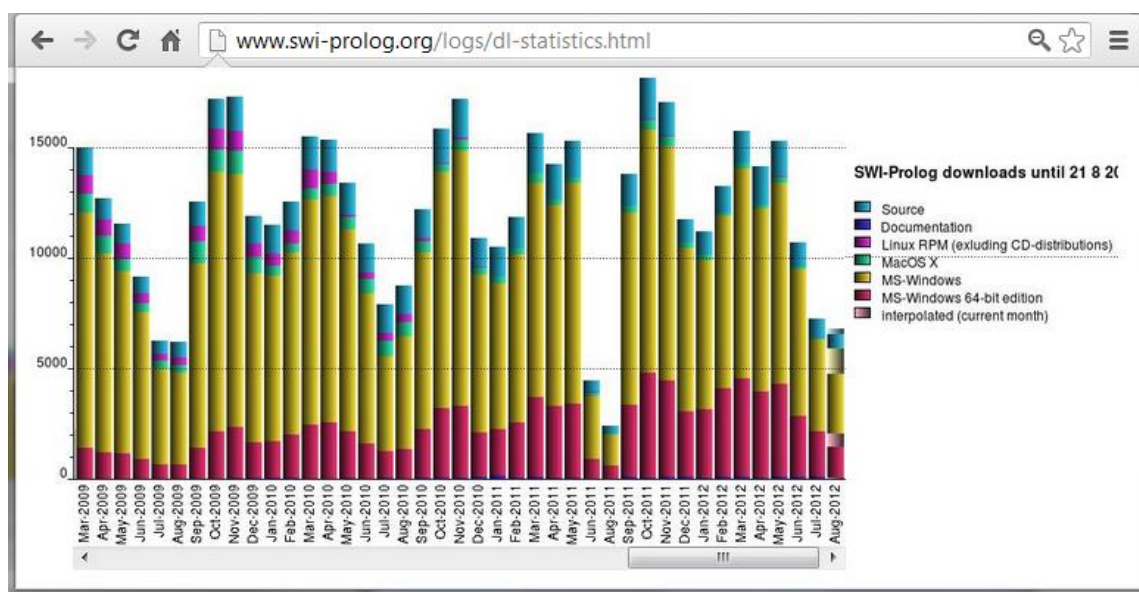


Рис. 1.6. Статистика загрузок дистрибутива с сайта SWI-Prolog.

Это говорит о достаточной популярности этого продукта в мире. А если еще сравнить эти значения с количеством загрузок в 2001-2005 годах, то из этого сравнения будет очевидно, что количество пользователей SWI-Prolog за последнее время увеличилось почти в 2 раза. И это в то самое время, когда слышны высказывания о том, что Prolog уже умер. Может те,

кто это говорит, просто живут в другом мире – мире дизайнов веб-сайтов и бухгалтерских программ. Но ведь кто-то должен решать и сложные задачи контроля, диагностики, управления и принятия решений в технических, биологических и социальных системах.

Разработка SWI-Prolog проводится в университете города Амстердам, Нидерланды с 1987 года. Его создателем и основным разработчиком является Jan Wielemaker. Название SWI – это аббревиатура от голландского *Sociaal Wetenschappelijke Informatica* («Social Science Informatics»), первоначального названия научной группы университета, в которой работает Jan Wielemaker. Сейчас название этой группы сменилось на новое название HCS (Human Computer Studies).

SWI-Prolog почти как все реализации в основном следует знаменитой "эдинбургской версии", но содержит частично реализованные особенности ISO Prolog. К числу основных особенностей последних версий SWI-Prolog следует отнести:

- Богатые библиотеки предикатов.
- Возможность написания логических модулей для веб-приложений.
- Поддержка GUI для XPCЕ и PreEmacs.
- Встроенная командная строка.
- Работа с файлами.

SWI-Prolog содержит развитые средства разработчика, включая интегрированную среду разработки (англ. Integrated Development Environment - IDE) с графическими отладчиком и профилировщиком, и обширную документацию. Кроме базовых функций языка, платформа реализует многопоточность, юнит-тестирование, GUI, интерфейс к языку программирования Java, ODBC и т. д. Следует отметить, что SWI-Prolog имеет встроенный собственный веб-сервер и работает на Unix, Windows, Macintosh и Linux платформах.

Среди множества существующих реализаций Prolog'a в дальнейшем изложении будут рассмотрены наиболее распространенные версии, к которым можно отнести PIE, SWI-Prolog и SWI-Prolog Editor, Turbo Prolog и, наконец, Visual Prolog. Следует сразу оговориться, что рассматриваться будут не языковые реализации, а начальные навыки по работе в каждой из этих сред, особенности и возможности каждой из них. Для этих целей более чем достаточно тех начальных сведений о Prolog'e, как языке программирования, что приводятся в следующем разделе. Для подробного знакомства с самим языком логического программирования рекомендуется обратиться к источникам, которые указаны в разделе литературы и интернет-ресурсы данного пособия.

2. ОСНОВЫ РАЗРАБОТКИ PROLOG-ПРОГРАММ

2.1. Общие сведения о языке Prolog

Теоретической основой Prolog является раздел символьной логики, называемый исчислением предикатов. Предикат — это логическая функция, которая выражает некоторое отношение между своими аргументами и принимает значение "истина", если это отношение имеется, или "ложь", если оно отсутствует.

В отличие от процедурных языков, где программист должен описать процедуру решения шаг за шагом, при использовании Prolog достаточно описать задачу и основные правила ее решения. Он имеет ряд свойств, которыми не обладают процедурные языки, что делает его мощным средством логического программирования. К числу таких свойств относятся:

- механизм вывода с поиском и возвратом,
- встроенный механизм сопоставления с образцом,
- простая, но выразительная структура данных с возможностью ее изменения.

Язык Prolog отличается единообразием программ и данных. Данные и программы лишь две различные точки зрения на объекты Prolog. В единой базе данных можно свободно создавать и уничтожать отдельные элементы. Поскольку не существует различия между программой и данными, то можно менять программу во время ее работы. Естественным методом программирования является рекурсия. Prolog является декларативным языком. Это означает, что имея необходимые факты и правила, он может использовать дедуктивные выводы для решения задач программирования. Prolog-программа содержит описание проблемы, которое представляет собой набор логических утверждений в форме фактов, таких как, например

```
Иван любит Марью.  
Петр любит футбол.
```

или в форме правил, например,

```
Иван любит X, если Петр любит X  
(Иван любит тоже самое, что любит и Петр ).
```

На основе имеющихся правил и фактов Prolog делает выводы. Имея два вышеприведенных факта и одно правило, он придет к выводу, что «Иван

любит футбол». А если задать запрос: «Кто любит футбол?», то он найдет все решения этой задачи. При этом он автоматически управляет решением задачи, стараясь во время выполнения программы найти все возможные наборы значений, удовлетворяющие запросу. Поиск производится по всей базе данных, ранее введенной в систему.

Термин *база данных* используется при объединении набора фактов для совместного их использования при решении некоторой конкретной задачи. Если кроме фактов в описании предметной области содержатся еще и правила, то часто используют термин *база знаний*.

В языке Prolog используется метод поиска с возвратом, который позволяет ему в случае нахождения одного решения пересмотреть все сделанные предположения еще раз, чтобы определить, не приводит ли новое значение переменной к еще одному решению. Синтаксис Пролога очень короток и прост. Поэтому его проще освоить, чем запутанный синтаксис традиционных языков программирования.

2.2. Понятие факта, правила, запроса и процедуры

Программа на языке Prolog состоит из множества предложений (фраз). Каждое предложение может быть одного из трех типов: это либо факт, либо правило, либо запрос.

- Факт — это фраза без условий, утверждение о том, что соблюдается некоторое отношение. Он записывается как имя предиката, за которым следует заключенный в скобки список аргументов. Каждый факт должен заканчиваться точкой. Например:

```
likes('Иван', 'Марья').  
likes('Петр', 'футбол').
```

- Правило — это факт, истинность которого зависит от истинности других фактов. Состоит из головы и тела, разделенных знаком `:-`, который читается как «если» и соответствует импликации. Голова правила — это предикат, а тело правила — последовательность предикатов, разделенных запятыми. Правило должно заканчиваться точкой, а запятая в теле правила означает конъюнкцию (логическое И). Например:

```
likes('Иван', X) :- likes('Петр', X).
```

Интуитивный смысл правила состоит в том, что цель, являющаяся головой, будет истинной, если Пролог сможет показать, что все выражения (подцели) в теле правила являются истинными.

```
Голова :- Подцель1, Подцель2, ... , ПодцельN.
```

Переменные, входящие в предикат головы правила, квантифицируются универсально. Это означает, что правило будет истинным для любых термов, которые удовлетворяют подцелям правила. С другой стороны, переменные, которые входят только в тело правила, квантифицируются экзистенциально. Учитывая изложенное, приведенное ниже правило

```
good_student( Name ) :- student( Name , B ) , B > 4.
```

можно прочесть так:

```
Для любого Name, Name является хорошим студентом
    ЕСЛИ существует средний балл B, такой что
        Name является студентом со средним баллом B
        И средний балл B больше 4.
```

- **Запрос** — это последовательность предикатов, разделенных запятыми или точкой с запятой и завершающаяся точкой. На естественном языке запятая соответствует союзу «и», а на языке математической логики обозначает конъюнкцию. Точка с запятой соответствует союзу «или» и обозначает дизъюнкцию. Предикат запроса называется целью. Простые запросы, не содержащие никаких переменных, допускают лишь два возможных ответа: "true" или "false". В случае ответа "true" говорят, что запрос завершился успехом, цель достигнута. Использование переменных в запросах позволяет задавать более сложные вопросы. Например:

```
?- likes('Иван', 'футбол').
true.
?- likes(Who, 'Марья').
Who = 'Иван' .
?- likes(Man, 'футбол').
Man = 'Петр' ;
Man = 'Иван'.
?- likes(X, 'Марья'), likes(X, 'футбол').
X = 'Иван' .
```

Последний из приведенных выше запросов на естественном языке можно интерпретировать как «Кто любит Марью и футбол?». С точки зрения логики предикатов — это составной (сложный) запрос.

- **Процедура** — это несколько правил, заголовки которых содержат одинаковые предикаты. Так, например, два правила

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

реализуют процедуру нахождения наибольшего из двух чисел и использует одинаковый предикат max/3, вида max(число1, число2, макс_число).

Считается, что между всеми правилами одной процедуры неявно присутствует соединительный союз «или», то есть все правила процедуры дизъюнктивно связаны между собой.

2.3. Механизм сопоставления и поиска с возвратом

Особенность языка Prolog заключается в том, что он воспринимает в качестве программы некоторое описание задачи или базы знаний и сам производит логический вывод, а также поиск решения задач, используя при этом поиск с возвратом и унификацию. Эти понятия подробно рассмотрим в следующих разделах, а здесь ограничимся только общим представлением и примерами.

Унификация представляет собой процесс сопоставления цели с фактами и правилами базы знаний. Цель может быть согласована, если она может быть сопоставлена с заголовком какого-либо предложения базы. Для этого предикат цели и предикат заголовка предложения должны иметь одинаковое имя и арность. Рассмотрим этот процесс на примере запроса

```
?- likes(Man, 'футбол').
```

При решении этой задачи Prolog ставит перед собой цель доказать истинность предиката likes/2 и найти условия, при которых это возможно на имеющемся множестве фактов и правил. Приняв за цель предикат запроса, Prolog пытается найти в базе знаний предложения в заголовке которых есть предикат likes/2.

Поиск подходящего для сопоставления предложения ведется с самого начала базы. Таким предложением будет первый факт likes('Иван', 'Марья'), имя предиката которого и его арность совпадает с именем и арностью предиката цели.

```
likes(Man, 'футбол')           - Исходная Цель
likes('Иван', 'Марья')         - Сопоставляемое предложение
Унификация:      Man = 'Иван' и 'футбол' ≠ 'Марья'
Сопоставление:   неудача (fail) → false
```

Проводя сопоставление этих двух предикатов, Prolog терпит неудачу при унификации аргументов этих предикатов. Если еще свободную переменную Man он может унифицировать с константой 'Иван', то вторые аргументы предиката, а именно, две константы 'футбол' и 'Марья' ему унифицировать не удастся.

Сопоставление цели с первым предложением базы знаний признается неудачным и Prolog откатывается к исходной цели. Он ищет в базе знаний новое предложение с заголовком likes/2 для выполнения сопоставления.

<code>likes(Man, 'футбол')</code>	- Исходная Цель
<code>likes('Петр', 'футбол')</code>	- Сопоставляемое предложение
Унификация:	<code>Man = 'Петр'</code> и <code>'футбол' = 'футбол'</code>
Сопоставление:	истина \rightarrow true

Таким предложением будет второй факт `likes('Петр', 'футбол')`. Это сопоставление Prolog признает успешным при условии, что свободная переменная `Man` будет унифицирована символьной константой `'Петр'`. Именно это значение и выводит Prolog в своей консоли как один из ответов на запрос к базе знаний.

Но Prolog еще не закончил просмотр всей базы знаний. Поэтому он выполняет откат к исходной цели и ищет еще какие-либо предложения для согласования. Таким предложением является заголовок правила

```
likes('Иван', X) :- likes('Петр', X).
```

которое можно согласовать с исходной целью при значениях свободных переменных `Man = 'Иван'` и `X = 'футбол'` и при условии, что тело правила, то есть его правая часть будет истинной.

<code>likes(Man, 'футбол')</code>	- Исходная Цель
<code>likes('Иван', X)</code>	- Сопоставляемое предложение
Унификация:	<code>Man = 'Иван'</code> и <code>'футбол' = X</code>
Сопоставление:	<code>likes('Петр', X)</code>
Новая цель:	<code>likes('Петр', 'футбол')</code>

Prolog в качестве новой цели принимает первую подцель тела правила, а именно предикат `likes('Петр', 'футбол')` и пытается доказать его истинность на множестве предложений базы знаний, начиная опять с первого факта. Весь рассмотренный выше процесс будет повторяться до тех пор, пока все возможные решения не будут найдены или процесс закончится неудачей. В системе Prolog для этого реализован метод решения, который называется поиском с возвратом.

Поиск с возвратом (англ. Backtracking) — это общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов на некотором множестве. Мы не будем математически строго останавливаться на рассмотрение этого метода, а поясним его работу, используя некоторую аналогию.

Очень часто метод поиска с возвратом сравнивают с поиском выхода из лабиринта. Нужно войти в лабиринт и на каждой развилке сворачивать влево, до тех пор, пока не найдется выход или тупик. Если впереди оказался тупик, нужно вернуться к последней развилке и свернуть направо, затем снова проверять все левые пути. В конце концов, выход (если он есть) будет найден. Подобным образом работает и механизм поиска с возвратом в

языке Prolog. Благодаря ему Prolog в состоянии находить все возможные решения, имеющиеся для данной задачи

При сопоставлении цели с фактом, цель согласуется немедленно. Если же сопоставление происходит с заголовком правила, то цель согласуется только тогда, когда будет согласована каждая подцель в теле этого правила, после вызова ее в качестве цели. При попытке повторного согласования система возобновляет просмотр базы с предложения, непосредственно следующего за тем, которое обеспечивало согласование цели ранее.

2.4. Основные элементы языка Prolog

Аргументы предложений Prolog-программы называются термами, а саму Prolog-программу можно рассматривать как сеть отношений, существующих между термами.

Каждый терм обозначает некоторый объект предметной области и записывается как последовательность литер, которые делятся на четыре категории: прописные буквы, строчные буквы, цифры и спецзнаки.

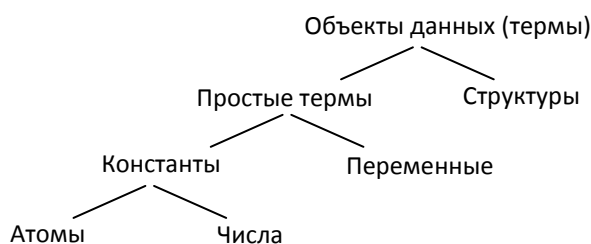


Рис. 2.1. Основные объекты данных языка Prolog.

Стандартный Prolog распознает тип объекта по его синтаксической форме и не требует какую либо дополнительную информацию (наподобие объявления типа данных) для того, чтобы распознать тип объекта. В качестве основных объектов данных (рис. 2.1) обычно выделяет три типа термов: константы, переменные (простые термы) и структуры (составные термы).

- Константа — поименованный конкретный объект или конкретное отношение. Существует два вида констант - атомы и числа. Атом - это либо последовательность латинских букв, цифр и знака подчеркивания, начинающаяся со строчной буквы, либо произвольная группа символов, заключенная в апострофы (одинарные кавычки). Иногда допускается использование и двойных кавычек. Примеры записи констант: `ivan`, `'Ivan'`, `invoice_n`, `"иван"`, `'№_счета'`, `'Иван'`.

Числа в Prolog бывают целыми (1, 1024, 0, -97) и вещественными (3.14, -0.0035, -0.5e4). Вещественные числа используются редко. Причина в том, что язык предназначен в первую очередь для обработки символьной, а не

числовой информации. При символьной обработке используют целые числа, например, для подсчета количества элементов списка, а нужда в вещественных числах очень невелика.

- Переменная — обозначение объекта, значения которого меняется в ходе выполнения программы. В Prolog она имеет имя, начинающееся с прописной буквы или знака подчеркивания. Например: Name, X, Invoice_n. Переменная называется связанной, если имеется объект, который она обозначает. При отсутствии такого объекта переменная называется свободной. Для обозначения переменной, на которую отсутствует ссылка в программе, используется анонимная переменная, которая обозначается одиночным знаком подчеркивания «_». Область действия любой из переменных — это предложение. Одноименные переменные в разных предложениях могут иметь разные значения.

- Список — объект, содержащий внутри произвольное число других объектов, упорядоченная последовательность элементов произвольной длины. Он задается перечислением элементов через запятую в квадратных скобках. Например:

[1, 2, 3, 4]	% список числ
[a, b, c]	% список констант
["Aa", "Bb", "Cc"]	% список строк
[1, a, "Aa", [22, 33, 44]]	% смешанный список
[]	% пустой список

Список является рекурсивным составным объектом. Он состоит из двух частей: головы списка, которым является первый элемент, и хвоста списка, который включает все следующие элементы.

```
?- A=[январь,31], [февраль,28], [март,31]] , A=[X|Y].  
A = [[январь, 31], [февраль, 28], [март, 31]],  
X = [январь, 31],  
Y = [[февраль, 28], [март, 31]].
```

- Структура (составной терм) — объект, состоящий из совокупности других объектов, которые называются компонентами. Структура в Prolog записывается с помощью указания ее функтора и компонент. Компоненты заключаются в круглые скобки и разделяются запятыми. Функтор записывается перед скобками. Компоненты сами являются термами. Например, в факт likes входит структура book(,_):

```
likes ( "Иван" , book ( "название", "автор" ) ).
```

Составные термы аналогичны записям Паскаля или структурам Си. Они представляют собой определяемые программистами объекты произвольной сложности.

- Предикат — логическая функция одного или нескольких аргументов, принимающая значение - истина или ложь. Записывается в виде составного терма. Аргументы перечисляются через запятую и представляют собой какие-то объекты или свойства объектов, а имя предиката обозначает связь или отношение между аргументами. Предикат однозначно определяется парой: имя и количество аргументов. Количество параметров предиката называют его арностью (arity) и указывают через «/» после его имени.

```
likes(Who, What).  
likes(Who, What, When).
```

Два предиката с одинаковым именем, но с разным числом аргументов, считаются различными. Например, предикаты likes/2 и likes/3 – это два различных предиката. Следует особо отметить, что имена предикатов записываются строчными буквами и не допускают пробелов. Поэтому в именах предикатов в качестве разделителей можно использовать символ подчеркивания.

Заканчивая разговор об основных элементах языка Prolog, отметим, что все они в той или иной степени составляет основу формирования фактов и правил – основных конструкций Prolog-программ. Что касается правил, то при их формировании для описания одного и того же знания могут быть использованы различные способы описания этого знания.

Например, всем известно, что бабушка человека - это мама его мамы или мама его папы. Но для описания этого знания возможны различные конструкции правил:

```
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)).
```

Выбор той или иной конструкции зависит не только от вкуса человека, разрабатывающего код, но и от его квалификации, знания особенностей работы конкретной Prolog-среды. Однотипные действия, реализованные в

разных конструкциях могут приводить к различным затратам ресурсов компьютера и скорости выполнения программы.

Эти замечание, приведенные в главе об основных элементах языка, имели своей целью только обратить ваше внимание на то, что для профессиональной работы необходимо не только знания об основных элементах и конструкциях языка, но и об реализованных взаимодействиях между ними в той среде, с которой вы работаете. Но об этом значительно позднее, а пока переходим к знакомству со средами разработки и запуска программ.

3. СРЕДА РАЗРАБОТКИ ПРОГРАММ PIE И СИСТЕМА SWI-PROLOG

Знакомство с Prolog'м лучше всего начинать с интерпретатора, который более или менее совместим с Edinburgh Prolog. К числу таких реализаций относятся PIE и SWI-Prolog. Среда PIE (Prolog Inference Engine) — это пример программы, написанной на Visual Prolog'e, которая включена в его дистрибутив поставки. PIE — это классический интерпретатор Prolog. С его помощью можно изучать Prolog и экспериментировать, не заботясь о классах, типах и других расширенных возможностях Visual Prolog.

Краткий учебник по работе с PIE есть на странице ее разработчика: [#PIE :_Prolog_Inference_Engine](http://wiki.visual-prolog.com/index.php?title=Fundamental_Prolog_Part_1). Архив есть на сайте <http://www.bitwisemag.com/apps/pie/pie.zip>. Для старта надо распаковать архив и запустить Pie.exe.

PIE представляет собой простую и легкую в использовании систему, которая имеет простой, стандартный для Windows, стиль многооконного пользовательского интерфейса, а также встроенный текстовый редактор с подсветкой синтаксиса (рис. 3.1).

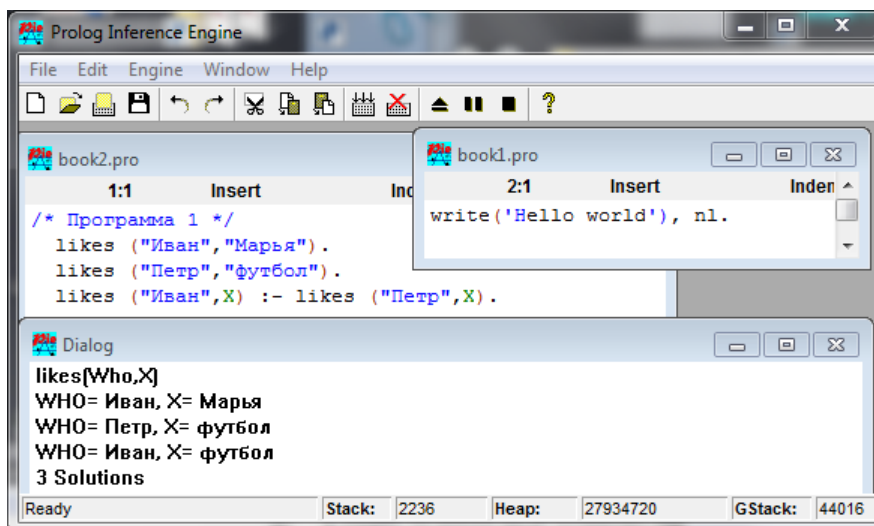


Рис. 3.1. Графический интерфейс среды PIE.

Из двух отмеченных выше систем, PIE проще в использовании для новичков. Однако, он достаточно прост, имеет небольшое число доступных языковых конструкций (предикатов). В нем нет средств визуального дизайна или отладки. Итак, если вы хотите пойти дальше в изучении Prolog,

то можно рекомендовать и установить более мощную реализацию Prolog, такую, например, как SWI-Prolog.

SWI-Prolog позволяет разрабатывать различные приложения, включая Web-приложения и параллельные вычисления. Сам SWI-Prolog — это консольное приложение, но в его дистрибутив входят GUI тулkit XPCE и встроенный в среду текстовый редактор PseEmacs. Он поддерживает автоматические отступы, подсветку и проверку синтаксиса, отладку и многое другое (рис. 3.2).

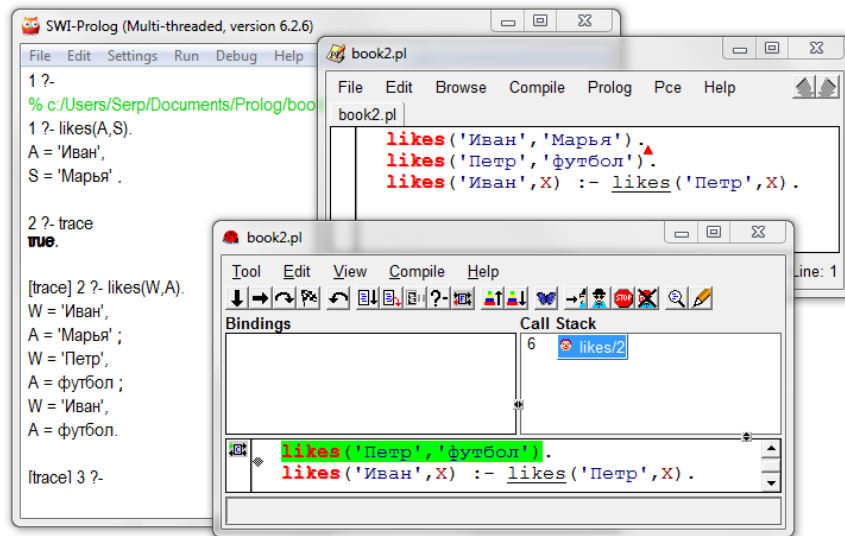


Рис. 3.2. Консоль SWI-Prolog, окна редактора и отладчика интерфейса PseEmacs.

Однако на начальном этапе работы с Prolog представляется более целесообразным работа в более простой среде SWI-Prolog-Editor. Это среда программирования для SWI-Prolog, включающая редактор программ с подсветкой синтаксиса, интерпретатор и отладчик программ (рис. 3.3). Основным назначением этой среды является обучение логическому программированию на языке Prolog.

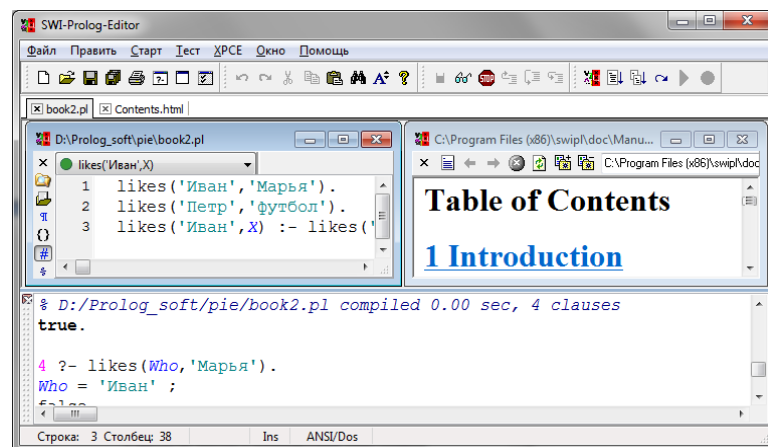


Рис. 3.3. Графический интерфейс среды SWI-Prolog-Editor.

Последние версии SWI-Prolog для MS Windows и MacOS доступны на сайте <http://www.swi-prolog.org>. Руссифицированную версию SWI-Prolog-Editor для MS Windows можно найти по адресу: <http://lakk.bildung.hessen.de/netzwerk/faecher/informatik/swiprolog/indexe.html>

Установка как SWI-Prolog, так и SWI-Prolog-Editor выполняется по стандартной для Windows процедуре. После установки и запуска SWI-Prolog-Editor необходимо выбрать пункт меню Окно -> Конфигурация, перейти на вкладку Настройки, в поле Codepage ввести 1251 (рис. 3.4).

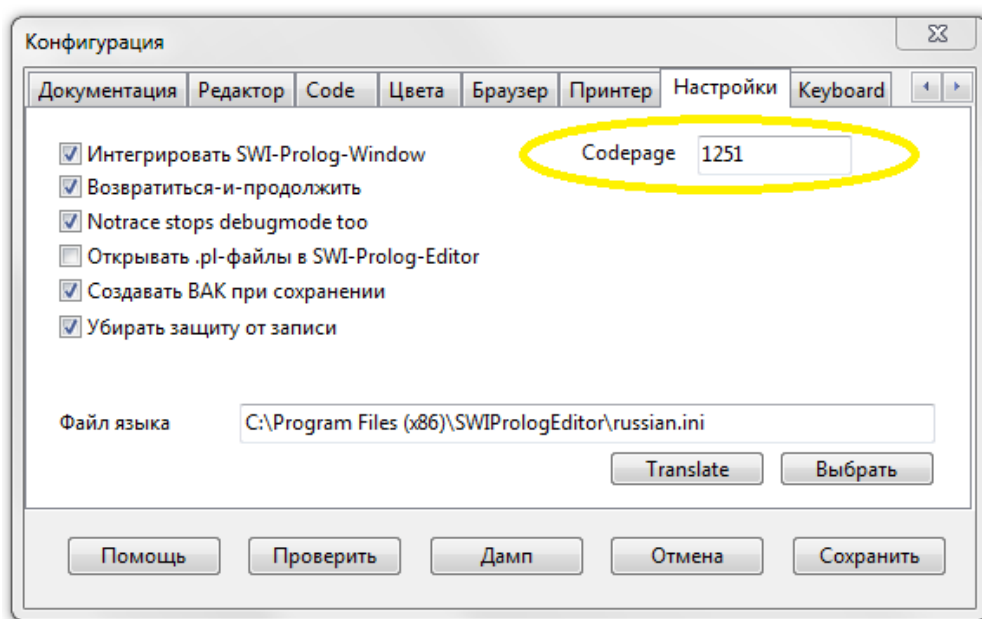


Рис. 3.4. Настройка кодовой страницы в SWI-Prolog-Editor.

Настройка кодовой страницы необходима для правильного сопоставления строковых констант, набранных русским алфавитом, между текстом программы в среде SWI-Prolog-Editor и языком SWI-Prolog.

3.1. Основы работы в консоли Prolog

Лучшим способом познакомиться как с описанными выше средами, так и с языком Prolog будет начало работы в этих средах, что даст возможность увидеть работу Prolog в действии. Это знакомство мы начнем с простейших действий и команд, выполняемых в консоли Prolog. Она в различных средах носит разные название: окно диалога, окно целей, окно запросов и тому подобное. Но суть ее остается неизменной. Консоль — это то место среды, куда пользователь может вводить команды, директивы или запросы. Однако полноценные запросы возможны только в том случае, когда в среду уже была загружена Prolog-программа или база знаний. Но

пока у нас нет никаких программ, то начнем с традиционного 'Hello World'. Для этого следует загрузить любую из выбранных вами сред разработки и ввести простую команду:

```
| write('Hello world'),nl.
```

Ввод можно выполнить либо в консоли SWI-Prolog после символов «?-» или в пустой строке в окне Dialog среды PIE. Убедитесь, что в конце строки вы не забыли поставить точку и нажмите Enter. Большинство Prolog систем выдаст результат, аналогичный тому, что приведен на рис. 3.5.

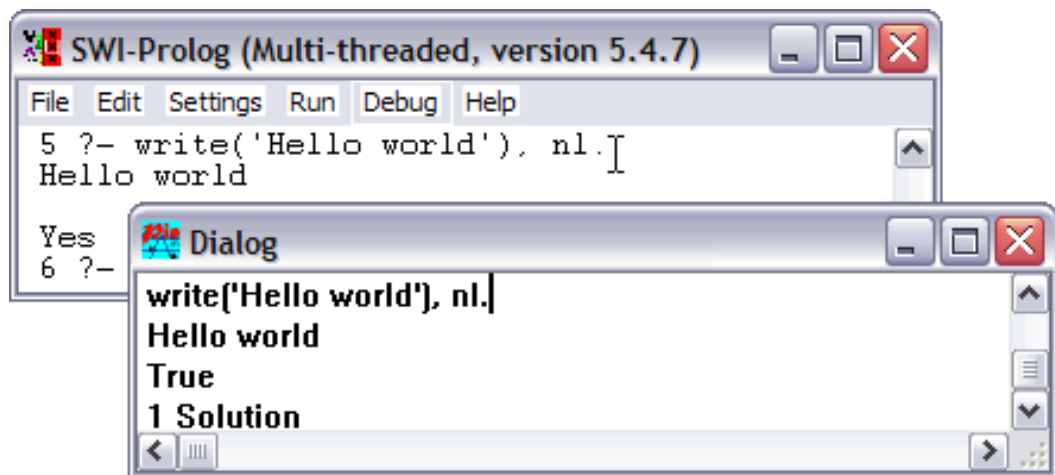


Рис. 3.5. Результат работы команды в среде SWI-Prolog и в среде PIE.

Первая строка вывода на экран является результатом работы введенных предикатов `write` и `nl`, а вторая строка — это результат оценивания запроса Prolog-интерпретатором. Значения `Yes` (в среде SWI-Prolog) и `True` (в среде PIE) указывает на то, что цель, которая была поставлена успешно решена. Кроме этого, PIE дополнительно еще указывает число решений, которые он нашел при ответе на данный запрос.

Предикат `nl/0` используется для вывода на экран символа перевода строки. Значение нуля после символа «/» в имени предиката указывает на то, что этот предикат не имеет аргументов и его арность равна нулю. Атом `'Hello world'` является аргументом предиката `write/1`. Этот предикат, как и предикат `nl/0`, относится к стандартным встроенным предикатам Prolog. Все они обычно описаны во встроенной справке. Так, в среде SWI-Prolog справку можно вызвать прямо из консоли, используя команды, например:

```
| ?- help.  
| ?- help(nl).  
| ?- help(write).
```

Традиционные реализации языка Prolog, такие как PIE и SWI, не являются строго типизированными. Это означает, что предикаты могут исполь-

зовать разные типы аргументов. В предикате `write/1`, например, аргумент может быть как числового, так и строкового типа. В этом можно убедиться, если ввести команду:

```
?- write(5), write(' * 3 = '), X is 5*3, write(X), nl.  
5 * 3 = 15  
X = 15.
```

Однако в этих двух средах есть и некоторые отличия в представления строковых данных. Так, PIE не различает одинарные и двойные кавычки в строковых константах и понимает их как атом. В SWI-Prolog замена в аргументе предиката `write/1` одинарных кавычек на двойные приводит к выводу строки в виде списка ASCII-кодов отдельных символов:

```
?- write("Hello, World!").  
[72,101,108,108,111,44,32,87,111,114,108,100,33]
```

В этом есть некоторое отличие SWI-Prolog от стандартной реализации, связанное с включением в его состав дополнительных средств обработки символьных данных и использования дополнительных типов данных. В частности, для рассматриваемого случая может быть рассмотрен пример, который иллюстрирует отмеченные выше замечания:

```
?- atom('Hello world').  
true.  
?- atom("Hello world").  
false.  
?- is_list("Hello world").  
true.
```

Эти команды используются встроенные предикаты SWI-Prolog `atom/1` и `is_list/1`, с помощью которых можно определить принадлежность аргумента к конкретному типу объекта данных. Среди термов языка Prolog есть и числа с которыми можно проводить арифметические действия. Но для осуществления арифметических действий требуется дополнительное указание. Например, следующий вопрос представляет собой наивную попытку потребовать выполнения арифметического вычисления:

```
?- X = 1 + 2.  
X = 1+2.
```

Как видно из примера, в ответ на Prolog ответил `X = 1 + 2`, а не `X = 3`, как мы хотели бы видеть. Причина заключается в том, что выражение `1 + 2` обозначает терм языка Prolog, в котором знак «+» является функтором, а значения 1 и 2 являются его компонентами. В запросе нет ничего такого, что вынудило бы систему Prolog активизировать операцию сложения. Для

определения этой проблемы предусмотрена специальная предопределенная операция `is`, которая вынуждает систему выполнить вычисление. Поэтому единственно правильный способ вызова арифметической операции состоит в том, чтобы команду представить в следующем виде:

```
?- X is 1 + 2.
X = 3.
```

При этом операция сложения была выполнена с помощью специальной процедуры, которая связана с операцией `is`. Подобные процедуры принято называть встроенными процедурами. Кроме того, в Prolog предусмотрены такие стандартные функции, как `sin(x)`, `cos(x)`, `atan(x)`, `log(x)`, `exp(x)` и т.д. Эти функции могут находиться справа от знака операции `is`.

Внимание!!!



- Ранее уже отмечалось, что в Prolog переменные начинаются с прописной буквы, а предикаты записываются строчными буквами.
- Аргументы предикатов, которые начинаются со строчных букв будут рассматриваться как атомы, а не как переменные.

Чтобы убедиться в зависимости аргумента предиката от того в каком регистре набрано его имя, достаточно ввести и выполнить следующую пару запросов:

```
?- X is 2, write(X),nl.
?- X is 2, write(x),nl.
```

Результатом первого запроса будет значение 2, а второго – `x`. Дело в том, что аргументом предиката `write/1` в первом случае является переменная `X`, а во втором случае – атом `x`. Рассмотрим еще ряд запросов, которые иллюстрируют различие в работе операторов `=` и `is`:

<pre>% среда SWI-Prolog ?- 4 is 2 + 2. true. ?- 4 is 2 + 3. false. ?- 4 = 2 + 2. false. ?- 4 = 4. true.</pre>	<pre>% среда PIE - это комментарий 4 is 2 + 2 True 1 Solution 4 is 2 + 3 No solutions 4 = 2 + 2 Unknown clause found 4 = 2 + 2 Execution terminated No solutions 4 = 4 True 1 Solution</pre>
--	---

Обратите внимание, что это оператор `is` оценивает арифметическое выражение до тестирования его на равенство. Оператор `=` не делает таких оценок и возвращает истину только в том случае, если обе стороны выражения были идентичны.

3.2. Основы работы с программными файлами

Оценить возможности Prolog по автоматическому логическому выводу при ответах на запросы можно только при использовании программных файлов, которые представляют собой базу знаний о какой-либо предметной области. Это, как правило, обычные текстовые файлы с расширениями `*.pro` (PIE и TurboProlog) и `*.pl` (SWI-Prolog). Эти файлы можно создавать в любом внешнем текстовом редакторе или внутри среды Prolog.

Для создание нового программного файла следует выбрать `File -> New`, и в появившемся окне ввести текст программы на языке Prolog. Пусть наша первая программа будет иметь вид:

```
/* Программа 2.1 */
likes('Иван', 'Марья') .           % факт
likes('Петр', 'футбол') .         % факт
likes("Иван", X) :- likes("Петр", X) . % правило
```

Далее надо сохранить файл, например, в папке `D:\Prolog` под именем `prog_1.pro` или `prog_1.pl`, в зависимости от используемой среды. Чтобы была возможность обращаться к данной программе с запросами ее необходимо загрузить в память. Для этого надо:

- используя среду PIE и SWI-Prolog, выбрать `File -> Consult (Ctrl+F8)`;
- используя среду SWI-Prolog-Editor, выбрать `Старт -> Запустить (F9)` или нажать соответствующую кнопку на панели инструментов (📄↓).

Если программа загрузилась успешно и Prolog не обнаружил в ней ошибок, то в консоле выдается соответствующее сообщение (рис. 3.6) и появляется возможность формировать запросы к программе.

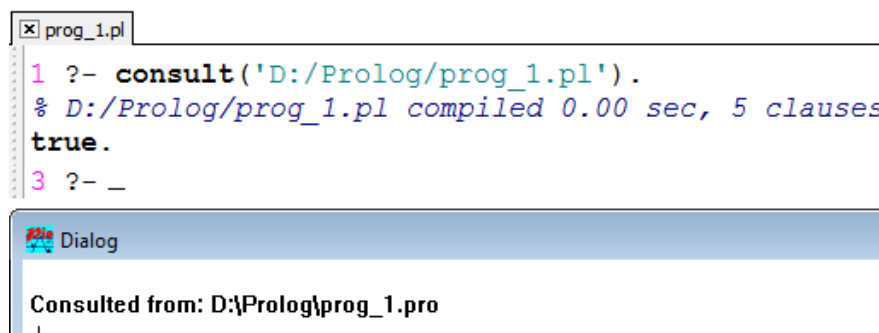


Рис. 3.6. Консоль SWI-Prolog и окно диалога PIE после загрузки программы.

Отличительной чертой SWI-Prolog является то, что запрос к программе набирается после «?-» и должен обязательно заканчиваться точкой. В PIE запрос можно вводить в любой строке. Более того, в окне диалога курсор можно перевести в строку ранее выполнявшегося запроса и, нажав клавишу Enter, запустить запрос на повторное выполнение. Повторное выполнение запроса в SWI-Prolog тоже возможно, если клавишами стрелка вверх и стрелка вниз выбрать нужный запрос, а затем нажать Enter.

Есть некоторые различия в формировании ответов на запрос в разных средах. В частности, если нас интересуют вопрос "Кто любит футбол", то запрос и ответ на него в среде PIE будет иметь вид:

```
likes(Who, 'футбол')
WHO= Петр
WHO= Иван
2 Solutions
```

Система выдала сразу все возможные ответы и сообщила о суммарном числе решений, которые она нашла при ответе на этот запрос. Если такой же запрос ввести в SWI-Prolog, то он выдаст только первый из возможных ответов и приостановит свою работу, ожидая реакции пользователя:

```
?- likes(Who, 'футбол') .
Who = 'Петр'
```

Прекратить выполнение программы, а точнее выдачу альтернативных ответов, можно, нажав клавишу «а», «с» или Enter (только в SWI-Prolog). Если возможны несколько вариантов ответа на запрос, то для получения каждого следующего используется «пробел», «;», «n», «r», или Enter (только в SWI-Prolog-Editor). Варианты ответов в панели SWI-Prolog отделяет друг от друга точкой с запятой «;».

```
?- likes(Who, 'футбол') .
Who = 'Петр' ;
Who = 'Иван' .
```

Запрос в SWI-Prolog можно набирать в несколько строк. Для перехода на новую строку используется клавиша Enter. Если строку закончить точкой и нажать Enter, то Prolog начнет выполнение запроса.

```
?- likes(X, 'Марья') ,
   |   write(X) ,
   |   write(' любит машу '), nl.
Иван любит машу

X = 'Иван' ;
```

Упражнение 3.1.



1. Выполнить все, приведенные в разделе 2.2, варианты запросов к этой программе. Модифицировать их с учетом последнего примера.
2. Сформировать и выполнить запрос о всех любителях чего-либо.

Разработанная программа может модифицироваться и добавляться, как и любая база знаний. В частности, если у пользователя появились новые знания о том, что “Все любители футбола посещают стадионы”, то это требует модификации исходной программы с тем, чтобы добавить в нее новое правило вида:

```
go(X,стадион) :- likes(X,футбол) .
```

После любой модификации программу надо снова загрузить в память, и если в ней нет ошибок, то Prolog в последней строке консоли выдаст true. Это позволит формировать новые запросы к программе. Например,

```
?- go(Fan,стадион) .  
Fan = 'Петр' ;  
Fan = 'Иван' .
```

Следует обратить особое внимание на то, что в тексте программы никаких фактов о посещении кем-то стадионов нет. Вместе с тем Prolog на основе встроенной процедуры логического вывода и на основе правил, описанных в программе, нашел всех возможных посетителей стадиона.

В программные файлы Prolog могут включаться не только базы знаний, описывающие некоторую предметную область, но и интерфейсные или управляющие процедуры. Элементарным примером такой процедуры может быть форматированный вывод внутренних запросов к программе, оформленных в виде некоторого целевого правила. Добавим в программу еще одно правило вида:

```
goal :- likes(A,S),write(A),  
          write(' любит '),write(S),nl,fail.
```

Сохраним и перезапустим программу, а затем введем запрос, который вызовет только что описанный предикат:

```
?- goal.  
Иван любит Марья  
Петр любит футбол  
Иван любит футбол  
false.
```

Из приведенного примера видно, что этот запрос выводит в более понятном для обычного пользователя виде всех любителей чего-либо, на основе данных, которые хранятся в программе.

Напомню, что цель данного параграфа знакомство с работой с файлами Prolog программ, а не изучение возможностей по формированию внешнего интерфейса. Цель последнего примера заключается только в том, чтобы показать, что предикаты и процедуры в Prolog-программе могут быть совершенно разного назначения. А коль это так, то встает вопрос, а какой смысл базу знаний и интерфейсные процедуры хранить в одном файле?

Упражнение 3.2.



1. Создать новый файл, перенести в него определение только одного предиката goal, сохранить файл под именем goal_1.
2. Файл основной программы без предиката goal сохранить в prog_1.
3. Перезапустить среду: Старт -> .Перезапустить (SWI-Prolog), закрыть оба файла (PIE).
4. Загрузить оба файла: Старт -> .Запустить все (SWI-Prolog), File -> Consult (PIE).
5. Ввести и выполнить запрос goal в окне консоли (SWI-Prolog) или окне Dialog (PIE).
6. Проверить работу всех ранее тестируемых запросов.

Цель данного упражнения состоит в том, чтобы сформировать два программных файла *.pl (или *.pro). Одновременно загрузить их в среду разработки Prolog и проверить доступность из консоли всех предикатов, описанных в обоих файлах. Это пример иллюстрирует возможность разбиения одной сложной задачи на ряд более простых. Если все было сделано правильно, то должно получиться нечто, аналогичное тому, что приведено на рис. 3.7.

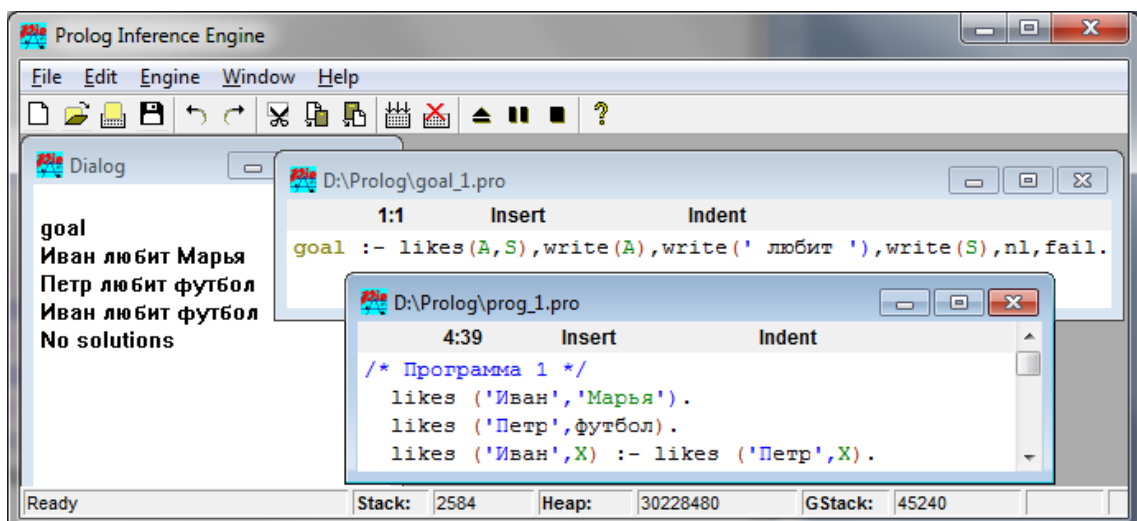


Рис. 3.7(а). Работа с двумя программными файлами в PIE.

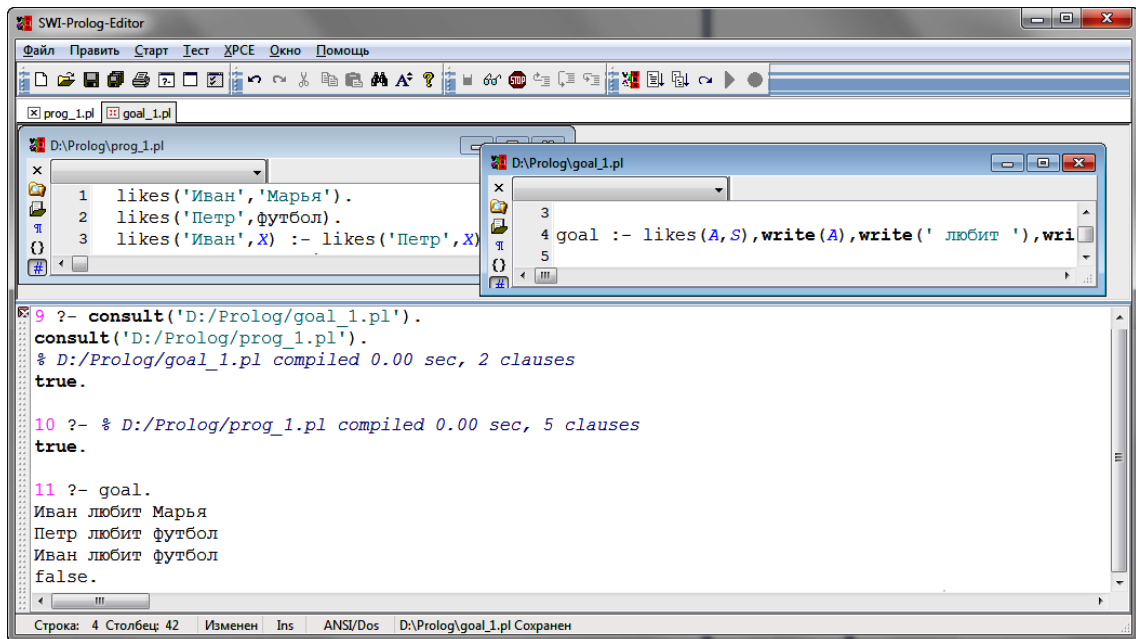


Рис. 3.7(б). Работа с двумя программными файлами в SWI-Prolog-Editor и PIE.

Если в PIE это единственный вариант декомпозиции задачи, то в SWI-Prolog, как более продвинутой версии, существуют более мощные средства формирования проектов, работы с модулями и подключение библиотек.

Остановимся еще на ряде аспектов работы с программными файлами из консоли SWI-Prolog. До этого момента мы открывали или загружали в среду Prolog программные файлы используя опции меню или кнопки панели инструментов. Вместе с тем, большую часть этих действий можно выполнить непосредственно из консоли, используя встроенные предикаты.

В частности, для того, чтобы указать рабочую папку, где хранятся исходные программные файлы Prolog, и которая должна быть текущей для конкретного сеанса работы, надо ввести команду:

```
?- chdir('D:/Prolog/').
```

Прямо из консоли можно узнать содержимое любой папки. Например, для того, чтобы выполнить просмотр содержимого текущей рабочей папки достаточно ввести команду:

```
?- ls.
goal_1.pl    goal_1.pro    prog_1.pl    prog_1.pro
```

Обнаружив в ней нужные вам программные файлы, можно перейти к их редактированию. Например, для вызова в SWI-Prolog встроенного в его среду текстового редактора PseEmacs с целью редактирования сразу двух файлов (рис. 3.8) достаточно в консоли ввести

```
?- edit(prog_1), edit(goal_1).
```

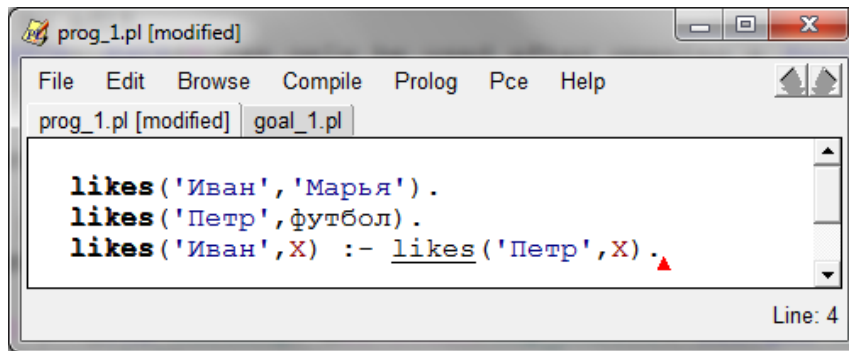


Рис. 3.8. Редактор PseEmacs с двумя закладками.

Для загрузки в среду Prolog сразу двух программных файлов можно воспользоваться одним из двух приведенных ниже вариантов:

```
?- [prog_1],[goal_1].
% prog_1 compiled 0.00 sec, 5 clauses
% goal_1 compiled 0.00 sec, 2 clauses
true.

?- consult('prog_1.pl'),consult('D:/Prolog/goal_1.pl').
% prog_1.pl compiled 0.00 sec, 5 clauses
% D:/Prolog/goal_1.pl compiled 0.00 sec, 2 clauses
true.
```

Результат выполнения любого из этих двух вариантов запросов будет аналогичен. В консоли Prolog выдаст информацию о том: какой файл загружен, сколько времени затрачено на его компиляцию, какое количество предложений обнаружено в компилируемом файле и каков результат выполненной загрузки каждого из файлов. При успешной компиляции в среде Prolog становятся доступны все факты и правила, описанные во всех этих программных файлах.

Рассмотрим еще один простейший вариант работы с базой знаний, которая построена на основе нескольких программных файлов. Этот подход базируется на том, что содержимое любого текстового файла или Prolog-программы может быть включено в другую программу на режиме ее загрузки. Для такой текстовой подстановки в SWI-Prolog используется предикат `include(имя_файла)`.

Если мы хотим, например, чтобы при загрузке файла `goal_1.pl` ему были бы доступны все предикаты из файла `prog_1.pl`, то для этого достаточно в первой строке файла `goal_1.pl` добавить строку:

```
:- include('D:/Prolog/prog_1.pl').
```

В том случае, когда был настроен путь к рабочей папке, либо путем выбора опций меню Старт -> Исходная папка, либо вводом в консоли пре-

диката `chdir('D:/Prolog/')`, первая строка `goal_1.pl` будет выглядеть еще проще, а именно:

```
:- include(prog_1).
```

После того, как файл `goal_1.pl` будет загружен и к нему будет выполнен запрос, вид среды SWI-Prolog_Editor должен быть аналогичен тому, что приведен на рис. 3.9.

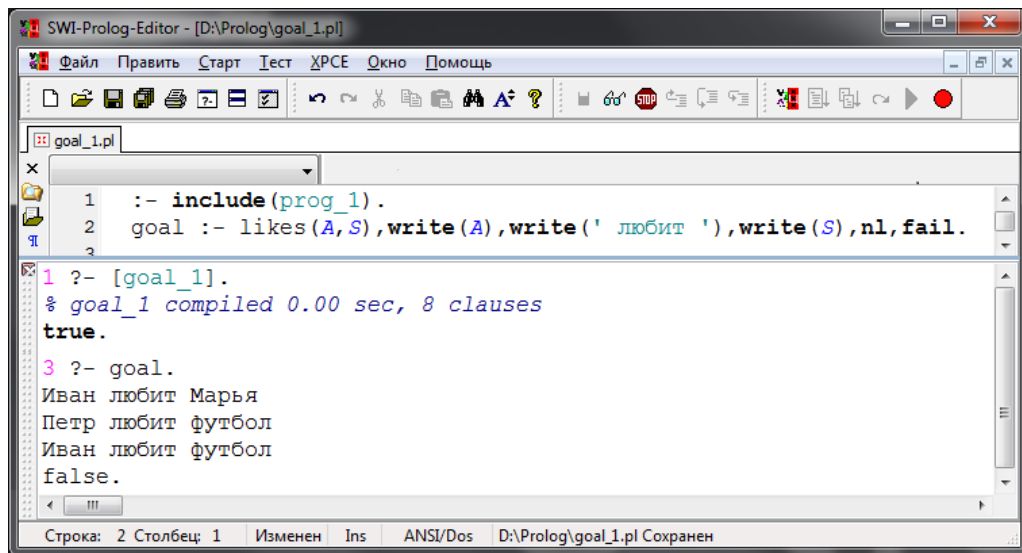


Рис. 3.9. Загрузка и работа с файлом `goal_1.pl`, использующим текстовую подстановку файла `prog_1.pl`.

Обратите внимание, что, в отличие от предыдущих подходов, в данном случае Prolog выполнил компиляция только одного файла, но который состоит аж из восьми предложений.

```
% goal_1 compiled 0.00 sec, 8 clauses
```

Но мы то хорошо знаем, что в самом файле `goal_1.pl` всего 2 строки. То есть, Prolog еще до начала компиляции в исходный файл подгрузил текстовое содержимое подключаемого файла, объединил все предикаты в один файл и уже компилировал этот объединенный файл. Убедиться в этом можно, если выполнить запросы к текущей базе знаний, используя предикаты, которые описаны в подгружаемом файле. Например,

```
?- likes('Петр',X).  
X = футбол.
```

Другими словами, используя в основном файле предикат `include` для тестовой подстановки из других программных файлов, мы после компиляции основного файла получаем доступную нам объединенную базу знаний.

Упражнение 3.3.



1. Создать из файла prog_1 новый файл prog_1_rus, в котором все предикаты, кроме встроенных, записаны на русском языке, а русские символьные константы заменены на соответствующие атомы.
2. Отладить программу prog_1_rus и выполнить к ней запросы, используя предикаты на русском языке.
3. Выполнить в основной программе тестовую подстановку сразу двух файлов prog_1 и prog_1_rus. Протестировать работу, вводя запрос на двух языках..

Заканчивая краткий экскурс в программные файлы Prolog, хотелось бы отметить, что хорошим стилем считается достаточно подробное описание содержимого каждого файла. Для этой цели используют комментарии, которые в Prolog-программе помещаются между символами «/*» и «*/» для любого числа строк, либо после символа «%» в той же строке.

3.3. Справка и помощь в среде SWI-Prolog

Одной из причин широкого распространения этой среды языка Prolog, особенно в учебных и образовательных целях, является то, что он достаточно хорошо документирован. В Интернете имеется большое количество учебных курсов, учебников и примеров программ. Правда, подавляющее их число расположено в англоязычной части Интернет, но и в русскоязычном пространстве за последние три года существенно увеличилось количество информационных источников.

Говоря о хорошей документированности SWI-Prolog и его клона SWI-Prolog-Editor, естественно, в первую очередь мы имеем в виду наличие внутренней справки. Она поставляется вместе с версией продукта и органично связана со средой программирования.

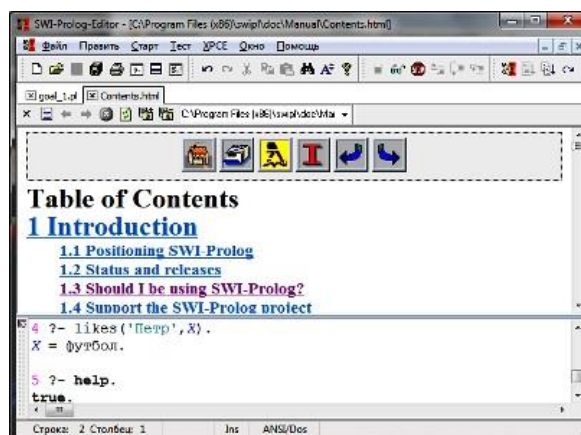


Рис. 3.10. SWI-Prolog-Editor и окно содержания Help.

Что касается среды SWI-Prolog-Editor, то в ней достаточно в основном меню выбрать опции Помощь -> Содержание, чтобы в верхней части интерфейса открылось еще одно окно с развитой навигацией по разделам описания SWI-Prolog (рис. 3.10).

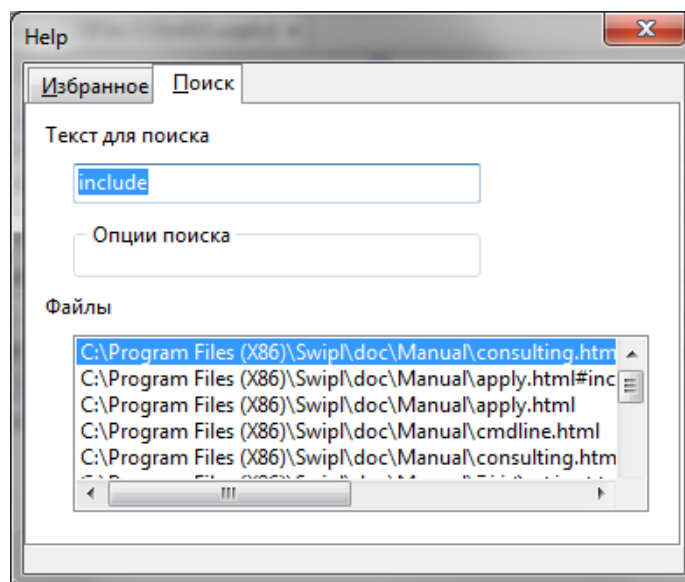


Рис. 3.11. SWI-Prolog-Editor - окно поиска по Help.

Если интересует конкретный предикат или конструкция языка Prolog, то удобнее использовать опцию меню Помощь --> Поиск (F1), которая позволит найти все разделы справки, где есть ссылки на эту конструкцию (рис. 3.11). Получить нужную информацию можно, если открыть соответствующий файл.

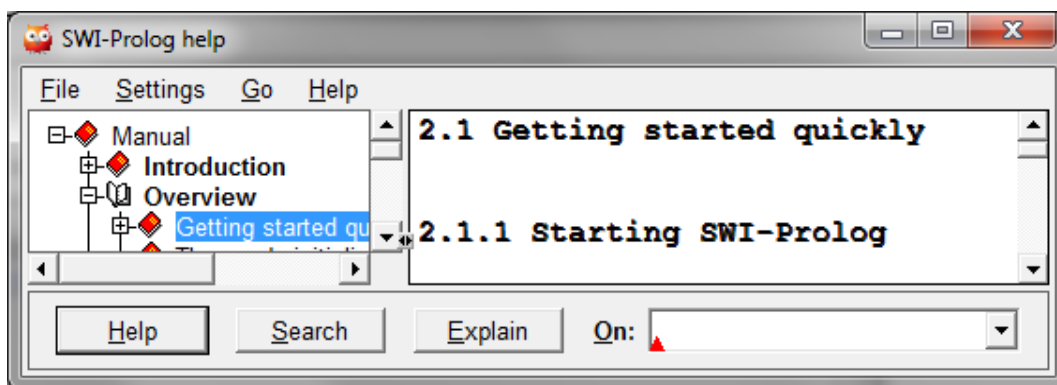


Рис. 3.12. Окно справки SWI-Prolog.

Наконец, в консоли в любой момент можно ввести `help(имя_предиката)` или просто `help`. Появится окно справки (рис. 3.12), в котором можно выбрать нужный раздел или выполнить поиск информации по конкретной конструкции SWI-Prolog'a.

3.4. Основы трассировки и отладки в среде PIE и SWI-Prolog

Трассировка программы предполагает пошаговую выдачу информации о выполнении предикатов и значениях унифицируемых переменных. Для включения трассировки надо использовать предикат `trace` или выбрать соответствующую опция главного меню. Трассировка программы в разных версиях Prolog выполняется одинаково, но может существенно различаться внешним интерфейсом. Так, в PIE доступна только текстовая трассировка, которую можно включить, выбрав опцию основного меню Engine -> Trace Calls. Эту же опцию используют и для отключения трассировки. Протокол трассировки после ввода запроса может иметь, например, такой вид:

```
Reconsulted from: D:\Prolog\prog_1.pro
Trace is On
likes('Иван',X) .
Trace: >> CALL:    likes(Иван,_)
Trace: >> RETURN:  likes(Иван,Марья)
X= Марья
Trace: >> REDO:    likes(Иван,Марья)
Trace: >> CALL:    likes(Петр,_)
Trace: >> RETURN:  likes(Петр,футбол)
Trace: >> RETURN:  likes(Иван,футбол)
X= футбол
Trace: >> REDO:    likes(Иван,футбол)
Trace: >> REDO:    likes(Петр,футбол)
Trace: >> FAIL:    likes(Петр,_)
Trace: >> FAIL:    likes(Иван,_)
2 Solutions
```

В процессе пошагового выполнения программы на экран построчно выдаются сообщения, которые содержат имя предиката и ключевое слово, определяющее текущее действие системы:

- **CALL:** – вызов предиката в качестве текущей цели.
- **RETURN:** – успех, управление передается вызвавшему его предикату.
- **FAIL:** – предикат не достиг успеха.
- **REDO:** - имеет место поиск с возвратом.

В отличие от PIE, в SWI-Prolog существуют два отладчика. Один из них традиционный – текстовый (консольный), а второй – графический. Что касается первого, то его протокол будет почти такой же, как и выше.

```
?- trace.
true.
```

```
[trace] 10 ?- likes(Who, футбол) .
    Call: (6) likes(_G495, футбол) ? creep
    Exit: (6) likes('Петр', футбол) ? creep
Who = 'Петр' ;
    Redo: (6) likes(_G495, футбол) ? creep
    Call: (7) likes('Петр', футбол) ? creep
    Exit: (7) likes('Петр', футбол) ? creep
    Exit: (6) likes('Иван', футбол) ? creep
Who = 'Иван' .

[trace] ?- notrace.
true.
```

Но существенным отличием будет то, что после выполнения каждого действия отладчик приостанавливает свою работу, обеспечивая пошаговое выполнение программы. При этом после выполнения каждого действия выводится знак вопроса, который соответствует запросу отладчика о той команде, которую он должен выполнить на этом шаге.

```
[trace] ?- likes(Who, футбол) .
    Call: (6) likes(_G495, футбол) ? h
```

Чтобы просмотреть все доступные команды отладчика, следует в ответ на вопрос ввести символ h (help) и на экране появится список всех возможных опций:

```
    Call: (6) likes(_G495, футбол) ? Options:
+:          spy          -:          no spy
/c|e|r|f|u|a goal: find  ..          repeat find
a:          abort        A:          alternatives
b:          break        c (ret, space): creep
[depth] d:  depth        e:          exit
f:          fail          [ndepth] g:  goals (backtrace)
h (?):      help          i:          ignore
l:          leap          L:          listing
n:          no debug      p:          print
r:          retry         s:          skip
u:          up            w:          write
m:          exception details
C:          toggle show context
```

Для того, чтобы выполнять программу по шагам (creep), как видно из приведенного списка, достаточно нажимать клавишу Enter или пробел в сообщениях, выводимых отладчиком,

- Call – означает прохождение узла дерева вывода сверху-вниз,
- Exit – означает прохождение узла дерева вывода снизу-вверх,
- а номер в скобках означает глубину узла в дереве вывода.

Однако на начальных этапах работы с программами на Prolog'е значительным преимуществом обладают графические отладчики, которые позволяют наглядно отследить процедуру вывода в конкретной программе для конкретного запроса. Для подключения графического отладчика можно либо в консоли ввести предикат `guitracer`

```
?- guitracer.  
% The graphical front-end will be used for subsequent  
tracing  
true.
```

либо установить опцию Тест -> GUITracer в основном меню SWI-Prolog-Editor'a или опцию Debug -> Graphical debugger в основном меню SWI-Prolog'a. Затем надо перевести систему в режим трассировки, либо выбрав опцию Тест -> Трассировка on/off, либо путем ввода в консоли `trace`.

После того как в консоли будет введен запрос, который требуется для трассировки работы программы, например,

```
?- trace.  
[trace] ?- go('Иван',S).
```

откроется окно графического отладчика (рис. 3.13), в котором панель Bindings отображает текущие значения переменных, Call Stack – состояние стека, т.е. глубину вложенности рекурсий, а нижняя – текст программы, где зеленым цветом выделяется выполняемый предикат.

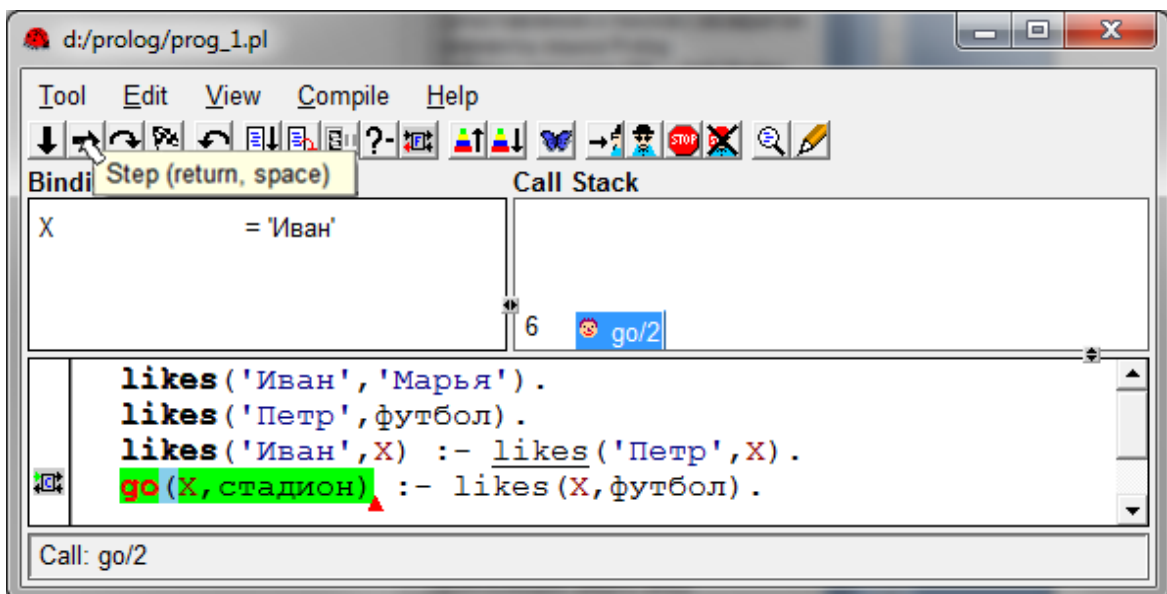


Рис. 3.13. Окно графического отладчика SWI-Prolog.

Для пошагового выполнения программы нужно нажимать на кнопку со стрелкой, либо клавишу пробел или Enter.

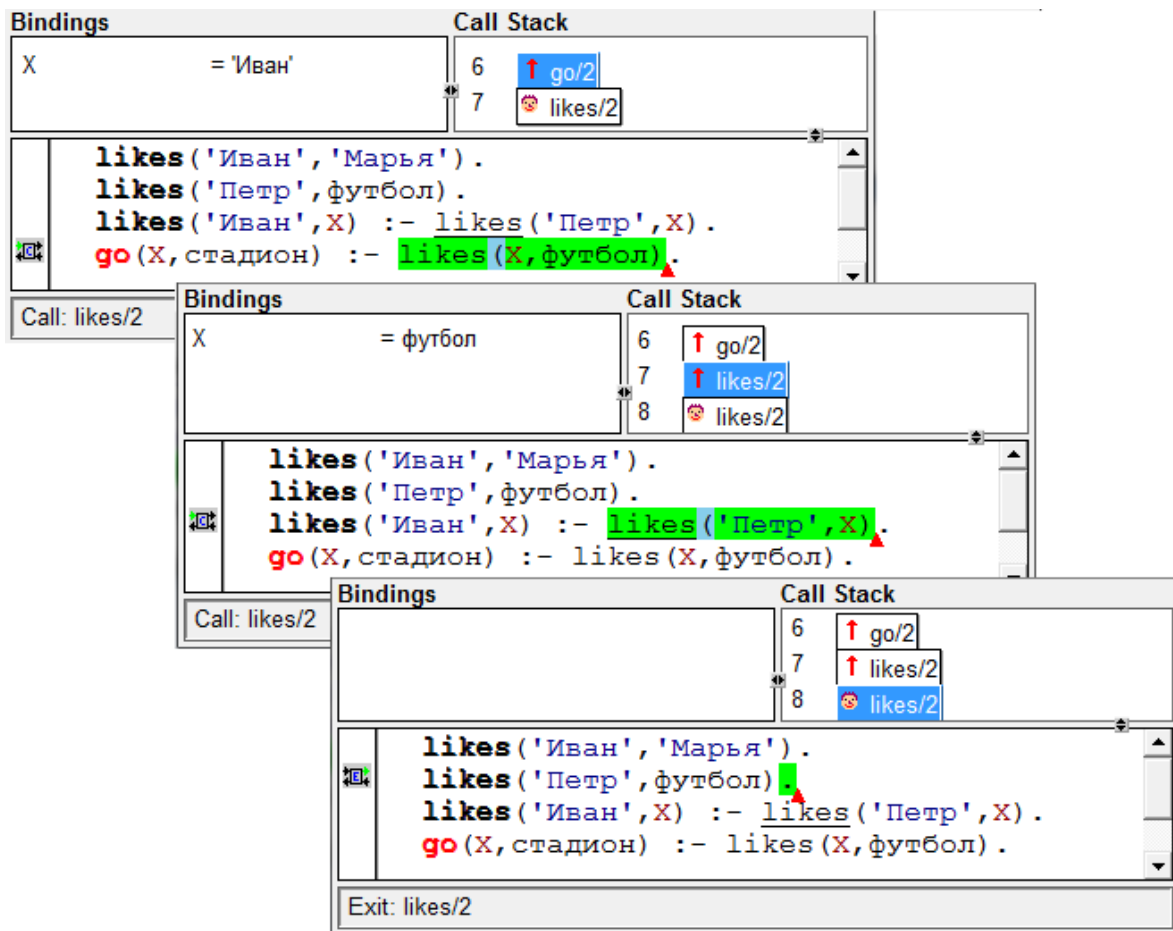


Рис. 3.14. Последовательность шагов трассировки программы.

Исходной целью является доказательство предиката `go(X,стадион)` с унифицированным значением переменной `X='Иван'`. Эта цель помещается в голову дерева решений – предикат `go/2` в панели Call Stack.

После того как будет инициирован первый шаг работы программы, для доказательства истинности исходной цели потребуется истинность тела правила, с которым унифицируется исходная цель. Для этого система формирует новую подцель `likes(X,футбол)` при `X='Иван'` и заносит ее в стек решений – `likes/2` на уровне 7 (рис. 3.14).

На втором шаге появится необходимость доказательства новой подцели `likes('Петр',X)` при `X=футбол`, которая также помещается в стек решений – узел `likes/2` на уровне 8. Эту подцель на третьем шаге системе удастся успешно согласовать на множестве предложений исходной задачи.

После этого на последующих шагах работы программы происходит обратный процесс, а именно, прохождение по узлам дерева вывода снизу-вверх – последовательному изъятию подцелей из стека с унификацией переменных (рис. 3.15).

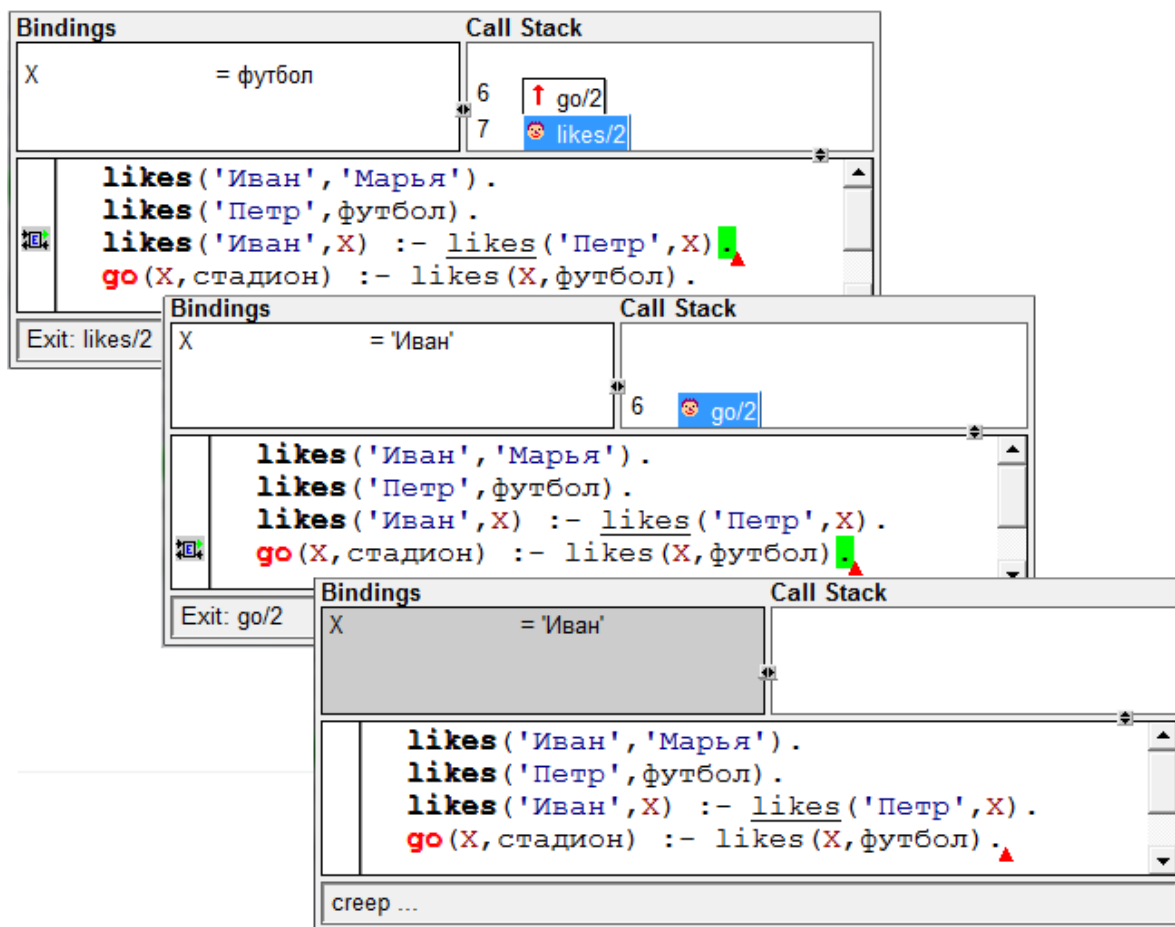


Рис. 3.15. Последовательность шагов трассировки вниз по дереву вывода.

Результат логического вывода будет представлен в консоли в следующем виде:

```
[trace] ?- go('Иван', S) .
S = стадион.
```

3.5. Графические интерфейсы на базе SWI-Prolog/XPCE

Язык Prolog, как правило, ассоциируется с искусственным интеллектом, базами данных и знаний, обработкой естественно языковых конструкций и другими подобными задачами. Реальные интерактивные программы могут включать в свой состав подзадачи, для которых Prolog является «the tool-to-use». Такие приложения часто реализуются на других языках, а Prolog используется в качестве встроенного движка для решения этих задач. Что же касается SWI-Prolog'a, то следует отметить, что в его дистрибутиве есть средство, которое позволяет разрабатывать, пусть и не очень сложный, но все же графический интерфейс пользователя. Таким средством в составе SWI-Prolog'a является XPCE.

XPCE – это платформо-независимый GUI тулkit для SWI-Prolog’a, Lisp и других интерактивных динамически типизированных языков. Хотя XPCE замышлялся, как не привязанный к конкретному языку программирования, наибольшую популярность этот фреймворк получил именно с Prolog’ом. Его развитие идет с 1987 года, с момента начала работ над SWI-Prolog’ом.

Являясь объектно-ориентированной библиотекой для разработки GUI, XPCE поддерживает окна, кнопки, меню, слайдеры, вкладки и другие базовые GUI виджеты. Ее разработчиками являются Anjo Anjewierden и Ян Wielemaker из отдела SWI Амстердамского университета.

Для того, чтобы ядро SWI-Prolog’a могло бы взаимодействовать с объектами XPCE и управлять ими из своей среды, в SWI-Prolog добавлены несколько предикатов, основными из которых являются:

- `new(?Reference, +Class(...Arg...))` – создает новый объект как экземпляр одного из классов XPCE (Class) с набором аргументов, используемых для инициализации объекта. Новому экземпляру объекта может быть присвоен указатель, доступный по ссылке (Reference) из SWI-Prolog’a.
- `send(+Reference, +Method(...Arg...))` – позволяет для объекта, которой указан ссылкой на него (Reference), вызвать нужный метод (Method) с требуемыми значениями аргументов.
- `get(+Reference, +Method(...Arg...), -Result)` – позволяет для объекта, указанного ссылкой (Reference), получить значения ряда параметров (Result), определенных в качестве аргументов (Arg) конкретного метода (Method).
- `free(+Reference)` – уничтожает объект, указанный ссылкой на него.

Естественно, что на начальном этапе работы со средой SWI-Prolog’a, мы не ставим перед собой задачу изучить классы, объекты и методы XPCE, а также способы взаимодействия с ними. Цель этого параграфа значительно скромнее. Она состоит только в том, чтобы показать имеющиеся в SWI-Prolog/XPCE возможности по разработке графических интерфейсов, хотя сила Prolog’a совсем в других областях использования.

С целью иллюстрации графических возможностей SWI-Prolog/XPCE рассмотрим небольшой пример по разработке простейшего графического интерфейса для программы 2.1.

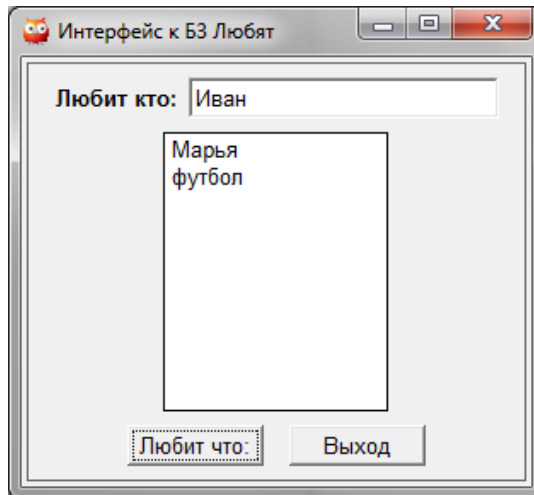


Рис. 3.16. Графический интерфейс к программе 2.1.

Как видно из рис. 3.16, интерфейс представляет собой окно диалога, на котором размещены четыре элемента управления: текстовое поле для ввода запроса, список – для вывода результатов и две кнопки. Одна запускает запрос на выполнение, а другая завершает программу. Один из возможных вариантов реализации этой задачи представлен в виде программы 2.2.

```

/* Программа 2.2 */
1 :- include('D:/Prolog/prog_1.pl').
2
3 gui_to_likes :-
4     new(MyWin, dialog('Интерфейс к БЗ Любят')),
5     send_list(MyWin, append, [
6         new(Who, text_item('Любит кто')),
7         new(MyList, list_browser),
8         button('Любит что:', message(@prolog,
9             output, MyList, Who?selection)),
10        button(выход, message(MyWin, destroy))
11    ]),
12    send(MyList, alignment, center),
13    send(MyWin, open(point(100,400))).
14
15 output(FrmList,X) :-
16     send(FrmList, clear),
17     likes(X,W),
18     send(FrmList, append, W),
19     fail.

```

В этой программе выполняется текстовое подключение исходной базы знаний (строка 1) и определяются два предиката: `gui_to_likes/0` и `output/2`.

Первый из них формирует экранную форму, второй – выполняет запрос и его результаты выводит в элемент управления список этой формы.

Определение предиката `gui_to_likes/0` включает в свой состав четыре соединенных запятыми предиката. Первый предикат (строка 4) – создает новый объект класса окна диалога, связывает ссылку на него с переменной `MyWin` и определяет текст заголовка этого окна.

Второй предикат (строки 5-11) – вызывает для объекта, ссылка на который определена в `MyWin`, метод `append` со списком вновь создаваемых объектов, определенных в XPCЕ классов.

Третий предикат (строка 12) – центрует положение на форме объекта, ссылка на который определена в `MyList`. Этот объект представляет собой элемент управления – список, созданный в строке 7 программы.

Четвертый предикат (строка 13) – отобразит на экране окно, которое было создано в памяти компьютера на предыдущих шагах программы, и размещает его на экране так, что левый верхний угол окна будет на экране иметь координаты `x=100` и `y=400`.

Несколько более подробно остановимся на списке вновь создаваемых объектов экранной формы, которые перечислены в методе `append` второго предиката:

- В строке 6 создается объект определенного в XPCЕ класса `text_item`. Это текстовое поле, позволяющее вводить информацию. При его определении указывается, что ссылка на этот объект сохраняется в переменной `Who`, что позволяет в дальнейшем по этой ссылке обращаться к объекту и вызывать его методы. Значение аргумента `label` этого объекта определяется как 'Любит кто'.
- В строке 7 создается объект класса `list_browser` (список). Ссылка на него сохраняется в `MyList`. Именно в этот объект будут выводиться результаты запроса.
- В строке 8 и 10 создаются объекты класса `button` (кнопка). Для них определены значения аргументов `label`, отображаемые в виде текста на кнопке, и обработчики событий, возникающих при нажатии на эти кнопки: `message(...)`.

Обработчик события `message(MyWin,destroy)` приводит к вызову метода `destroy` (уничтожение) для объекта указанного ссылкой `MyWin`. Так как с этой переменной у нас было связано диалоговое окно с размещенными на нем элементами управления, то это окно будет выгружено из памяти, что приведет к окончанию работы программы. Обработчик нажатия кнопки, вида

```
message(@prolog, output, MyList, Who?selection )),
```

имеет более сложную структуру, вызывая методы и аргументы объекта, на который указана ссылка в виде @prolog. Это абсолютная ссылка на объект, которым является собственно SWI-Prolog. Этим указано, что при обработке нажатия кнопки следует вызвать определенный в среде Prolog предикат output/2 и передать ему два аргумента: MyList – указатель на объект списка экранной формы и Who?selection – содержимое объекта, ссылка на который указана в переменной Who.

Что касается предиката output(FrmList,X), то его структура аналогична предикату goal из рассмотренных ранее примеров программ. Отличие в том, что вместо предиката write в описании output/2 используем предикат send(FrmList, append, W), который вызывает метод append с аргументом W для объекта FrmList. Но переменная FrmList при вызове output/2 из обработчика нажатия кнопки унифицирована переменной MyList. Точно также, как переменная X унифицирована значением Who?selection.

Это значит, что переменная FrmList в текущий момент содержит ссылку на объект список экранной формы, а значение переменной X содержит значение, которое было введено в поле ввода экранной формы. Поэтому вызов send(FrmList, clear) приведет к очистке списка на экранной форме, а после выполнения likes(X,W), добавлению в список на экранной форме значений переменной W с помощью предиката send(FrmList, append, W).

Упражнение 3.4.



1. Загрузите программу 2.2, исследуйте ее работу на разных запросах.
2. Измените местоположение и вид экранной формы программы.
3. Переделайте программу так, чтобы выполнялись запросы: "Любит что ... кто?".
4. Если хватит смекалки и знаний, объедините в одном интерфейсе возможность прямых и обратных запросов.
5. Все версии модификаций сохранять в рабочем каталога диска.

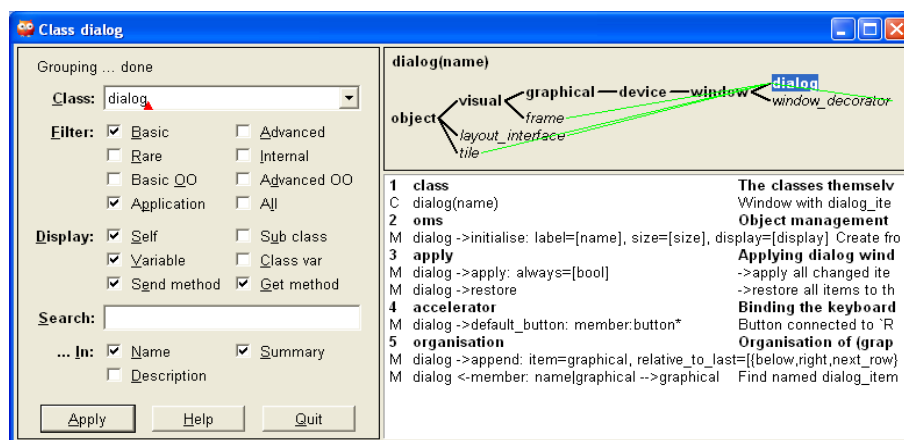


Рис. 3.17. Описание класса dialog в окне Class browser.

Следует отметить, что в XPCE включен достаточно большой набор элементов управления, которые могут быть использованы в окнах диалога. Не вдаваясь в подробности, только перечислим часть predefined классов XPCE: button, text item, int item, slider, menu, menu bar, label, list browser, editor, tab, tab stack, dialog group.

О назначении большинства из них можно догадаться уже по их названиям. Более подробно узнать об определенных в XPCE классах и их иерархии можно по справке, которая встроена в систему. Например, можно в главном меню SWI-Prolog-Editor выбрать опцию XPCE -> Class browser, а затем в поле со списком Class экранной формы ввести или выбрать из списка интересующий нас класс. Это позволит получить его описание и иерархию его определения (рис. 3.17).

Однако возможен и другой подход, при котором в главном меню SWI-Prolog-Editor выбираем опцию XPCE -> Class hierarchy. Далее, раскрывая дерево объектов, выбираем интересующий нас класс, правой кнопкой мыши вызываем всплывающее меню. В нем выбираем опцию Documentation для вызова документа с описанием этого класса (рис. 3.18).

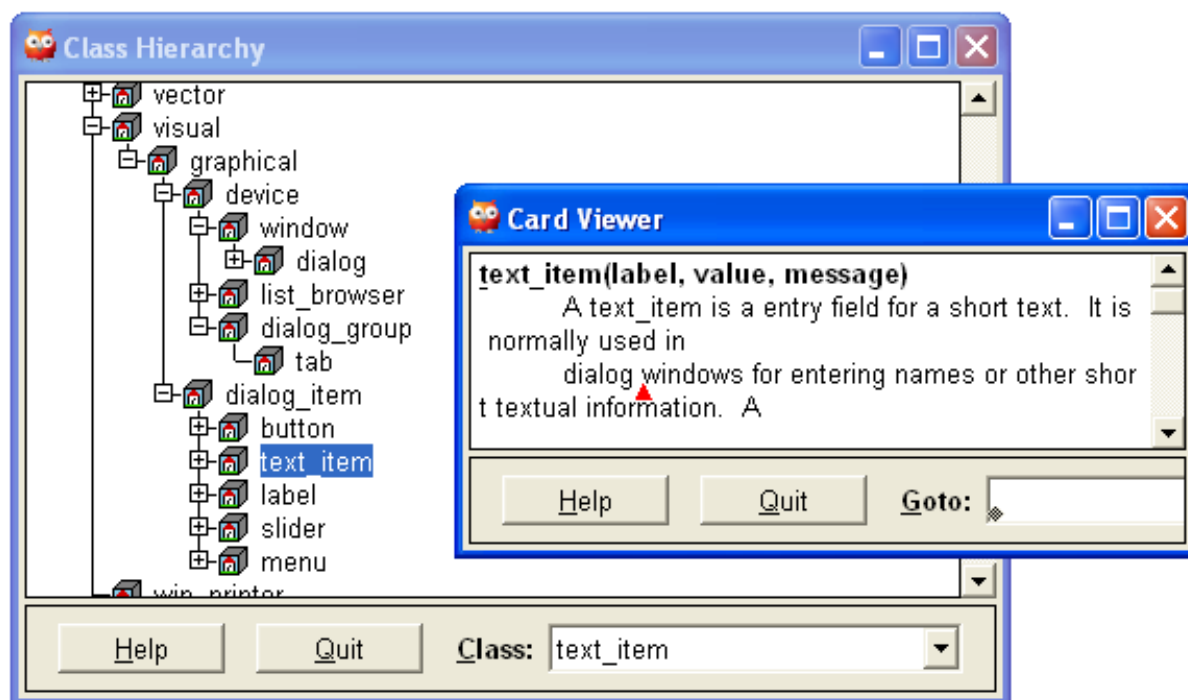


Рис. 3.18. Дерево классов и окно документации по text_item.

Построение экранных форм на основе окон диалога и компонентов класса dialog_item представляет собой достаточно простую задачу после получения некоторых навыков работы с ними. Этот подход удобен еще и тем, что для простых форм даже можно не указывать месторасположение элементов управления на форме. Это XPCE выполнит сам, но, если что-то

не устраивает, можно просто указать, что тот или иной компонент должен располагаться справа или слева от другого, или с новой строки.

Однако в ХРСЕ существует и более развитый набор графических средств, которые позволяют решать широкий класс графических задач. Для этого в ХРСЕ есть класс `picture`, который на экране отображается в виде окна с полосами прокрутки, а с точки зрения работы с ним разработчика – это практически бесконечная двумерная область, в любом месте которой может быть отображен тот или иной графический примитив. К основным таким примитивам ХРСЕ обычно относят: `arrow`, `bezier`, `bitmap`, `pixmap`, `box`, `circle`, `ellipse`, `arc`, `line`, `path`, `text`.

Для знакомства с возможностями ХРСЕ по разработке графических приложений рассмотрим два простейших примера, которые заимствованы из книги Jan Wielemaker, Anjo Anjewierden “Programming in ХРСЕ/Prolog”, но несколько модернизированы.

```
/* Программа 2.3 */
1 graf_test :-
2   new(Pict, picture('Вывод графических примитивов')),
3   send(Pict, open),
4   send(Pict, display, new(Box, box(100, 100))),
5   send(Pict, display, new(Cir, circle(50)), point(25, 25)),
6   send(Pict, display, new(BM, bitmap('folder.ico')),
7         point(100, 100) ),
8   send(Pict, display, new(Txt, text('Привет')),
9         point(120, 50) ),
10  send(Pict, display, new(BZ, bezier_curve(point(50, 100),
11                                           point(120, 132),
12                                           point(50, 160),
13                                           point(120, 200))) ),
14  send(Pict, display, new(Mes, text('Нажми Enter,
15                                   пожалуйста ... ')), point(220, 150) ),
16  get0(_), free(Mes),
17  send(Box, radius, 10),
18  send(Cir, fill_pattern, colour(orange)),
19  send(Txt, font, font(times, bold, 18)),
20  send(BZ, arrows, both).
```

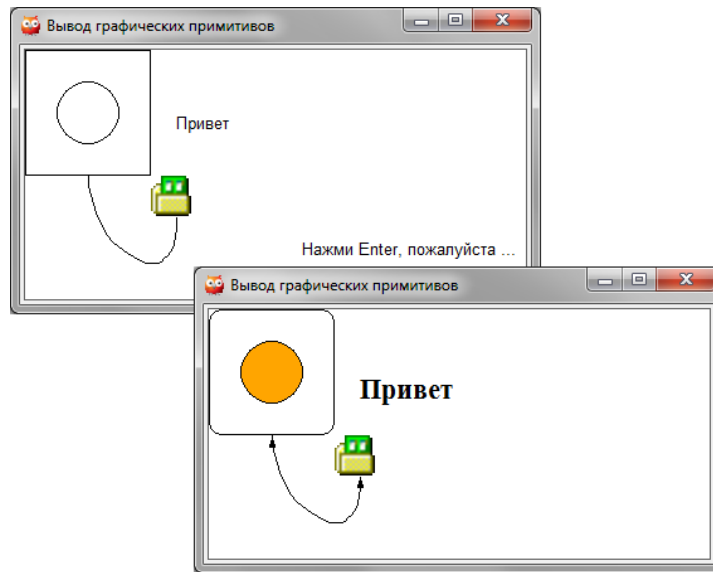


Рис. 3.19. Результат работы программы 2.3 до и после нажатия Enter.

Программа 2.3 состоит из двух частей. Их разделяет строка 16, где ожидается нажатие Enter. В первой части – создается новый объект класса `picture`, ссылка на который сохраняется в переменной `Pict`. Затем на этом объекте создаются различные графические примитивы с определенным набором свойств. Вторая часть программы показывает, как в процессе ее работы можно изменить свойства ранее созданных графических объектов или вообще удалить их.

Второй тестовый пример иллюстрирует совсем нетипичную для Prolog'a вычислительную задачу расчета и построения графика функции $\sin(x)$ в диапазоне от 0 до 360 градусов (рис. 3.20). Этот пример приводится исключительно в целях демонстрации графических возможностей XPCЕ.

```
/* Программа 2.4 */
draw_sin :-
    send(new(Pict, picture('Функция SIN(X)')), open),
    W is 400, H is 220,
    send(Pict, size, size(W,H)),
    send(Pict, display, line(0,H/2,W-20,H/2)),
    send(Pict, display, new(P, path(kind:=poly))),
%    send(Pict, display, new(P, path(kind:=smooth))),
    ( between(0, 360, X),
      Y is (H/2 - sin((X * 6.283185)/360) * 100),
      send(P, append, point(X, Y)),
      fail
    ; true
    ).
```

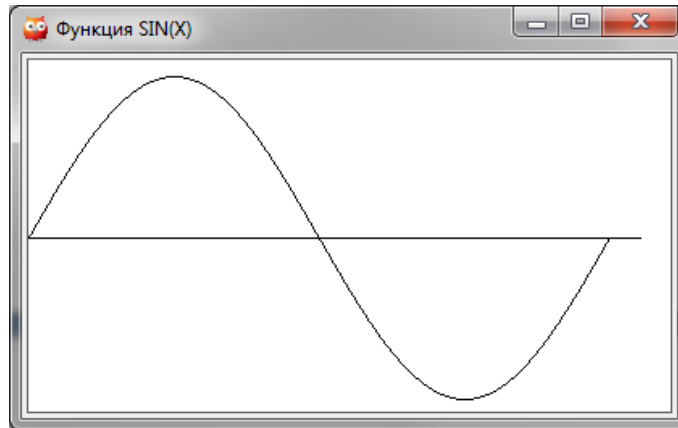


Рис. 3.20. Результат работы программы 2.3 до и после нажатия Enter.

Упражнение 3.5.



1. Загрузите программы 2.3 и 2.4, исследуйте их работу.
2. В программе 2.3 измените места объектов на форме, набор свойств и их значений, добавьте новые примитивы.
3. В программе 2.4 перенесите комментарий на строку выше и разберитесь – чем различаются значения `poly` и `smooth`.

3.6. Загрузка и запуск программ SWI-Prolog'a

До текущего момента мы все программы разрабатывали, запускали и исследовали в интерактивной среде SWI-Prolog'a или SWI-Prolog-Editor'a. При этом SWI-Prolog-Editor представляет собой интегрированную среду, которая обращается к SWI-Prolog'у и работать без него не может. Наступил момент, когда надо чуть подробнее взглянуть на SWI-Prolog. Причем, не как на среду разработки Prolog программ, а как на программный продукт.

Если обратиться к свойствам ярлыка рабочего стола или опции меню Windows, с помощью которых может быть запущен SWI-Prolog, то в поле командной строки вызова программы можно увидеть текст, аналогичный

```
"C:\Program Files (x86)\swipl\bin\swipl-win.exe" --win_app
```

Из него видно, где находится инсталляционный каталог SWI-Prolog'a. Если теперь войти в его подкаталог `\bin`, то там, наряду с большим количеством библиотек, можно найти два исполняемых файла, а именно:

`bin\swipl-win.exe` – это графическое приложение для интерактивного использования, которое устанавливается по умолчанию.

`bin\swipl.exe` – консольная версия для использования в сценариях.

В зависимости от того, в каком режиме предполагается использовать SWI-Prolog, надо запускать ту или иную программу. Формат командной строки вызова любой из этих версий практически аналогичен и в общем случае имеет вид:

```
swipl.exe <--опции> -f <файл.pl> -g <цель> -t <цель>
```

где swipl.exe – имя вызываемого файла, за которым следует набор опций и ключей. Опции определяют настройку и режим работы программы, а ключи – набор действий, выполняемых SWI-Prolog’ом по загрузке и ходу выполнения Prolog программ. Подробно об их составе и назначении можно узнать из справки по системе, а здесь рассмотрим только основные из них.

Среди всего множества опций на первых шагах отметим только две из них: --win_app и --quiet. Если SWI-Prolog запущен с опцией --win_app, то в качестве рабочего будет установлен каталог \MyDocuments\Prolog. Если его нет, то система создаст его заново. Использование при вызове SWI-Prolog’a опции --quiet отключает вывод начального сообщения системы в окне консоли при старте программы. Ключ -q выполняет то же действие.

Что касается ключей, которые указываются в строке вызова, то с их помощью можно указать SWI-Prolog’у на необходимость при старте загрузить и откомпилировать определенные программные файлы, а также выполнить запросы для инициализации ряда целей или управления системой. Наиболее часто используются следующие ключи: -f, -s, -g, -t, -q.

Ключи -f <файл> или -s <файл> указывают на необходимость при старте системы загрузить соответствующий программный файл или файлы. Если при старте системы необходимо загрузить сразу несколько файлов, то их имена перечисляются после ключа, объединяясь знаком «+». Например, swipl-win.exe -f d:\prolog\prog_1.pl + d:\prolog\prog_1_rus.pl.

Ключи -g <цель> и -t <цель> определяют цели, которые должны быть инициализированы при старте системы (-g) или после выполнения всех предикатов загруженных программных файлов (-t). Цель, которая указана после ключа может включать в себя несколько предикатов, перечисляемых через запятую. К числу наиболее часто используемых терминальных целей относится предикат halt, который вызывает завершение работы Prolog’a.

Использование строки запуска позволяет настроить среду SWI-Prolog’a для работы, но особенно это важно для программ с внешним интерфейсом. Когда мы работаем в интерактивной среде, то есть формируем прямо в ней файлы программ и вводим запросы в консоль, настройка строки запуска позволит ускорить переход от одной задачи к другой. При разработке приложения, которое будут использовать другие пользователи, и в котором предусматривается внешний интерфейс, то здесь без формирования строки запуска обойтись вообще затруднительно. Рассмотрим это на примерах.

3.6.1. Использование ярлыков для запуска программ

Ранее уже были разработаны две простейшие программы, которые можно рассматривать как пример текстового (goal_1.pl) и графического (prog_2_2.pl) интерфейсов по доступу к нашей тестовой базе знаний. Но чтобы работать с ними, пользователь должен знать как запустить SWI-Prolog, как и какие программные файлы следует загрузить и, наконец, как и какой предикат следует использовать для начала работы программы.

Избавиться от большей части этих проблем можно, если для запуска интерфейсных программ создать ярлыки с соответствующими командными строками. Для примера, создадим ярлык запуска программы с текстовым интерфейсом (рис. 3.21), и сохраним его под именем start_win_1 в той же папке, где находится и файл goal_1.pl.

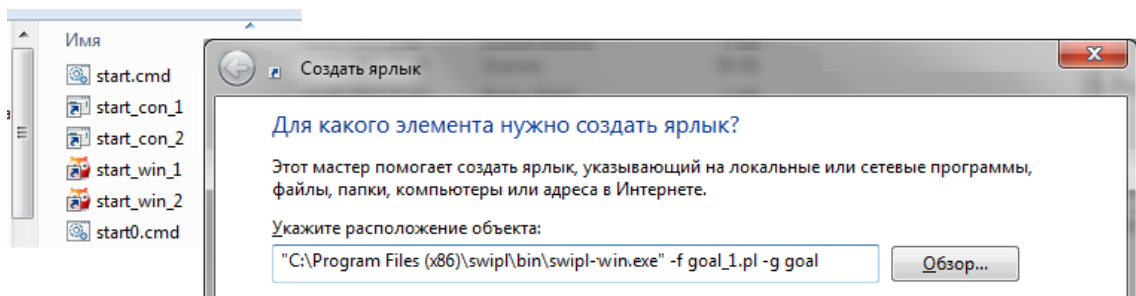


Рис. 3.21. Запуск SWI-Prolog с загрузкой goal_1.pl и стартовым предикатом goal.

Двойной щелчок мышкой по этому ярлыку даст результат, аналогичный тому, что приведен на рис. 3.22 слева. Если изменить строку запуска и добавить в нее ключ -q, то после его запуска результат будет аналогичен тому, что приведен на рис. 3.22 справа. Из сравнения этих окон видно, какое влияние на работу SWI-Prolog'a оказало введение ключа -q в строку его запуска. Аналогичный результат будет получен, если ключ -q заменить на опцию --quiet.

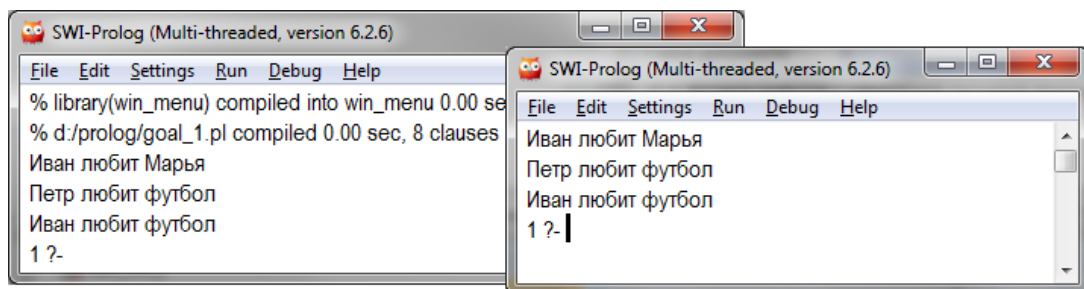


Рис. 3.22. Результат запуска ярлыка start_win_1.

Продолжим знакомство с возможностью запуска программ вне среды SWI-Prolog'a. Скопируем и переименуем ярлык start_win_1 в start_win_2, а

затем несколько модифицируем строку запуска, изменив в ее правой части ключ загрузки на другой файл:

```
|| . . . -s d:\prolog\prog_2_2.pl -g gui_to_likes
```

Вызов этого ярлыка приведет к мгновенному запуску оболочки нашего графического интерфейса программы по доступу к тестовой базе знаний (рис. 3.23). Появляется возможность вводить запросы и получать ответы, используя поля экранной формы. Однако при этом оказывается доступной и консоль SWI-Prolog'a, в которой также возможна работа.

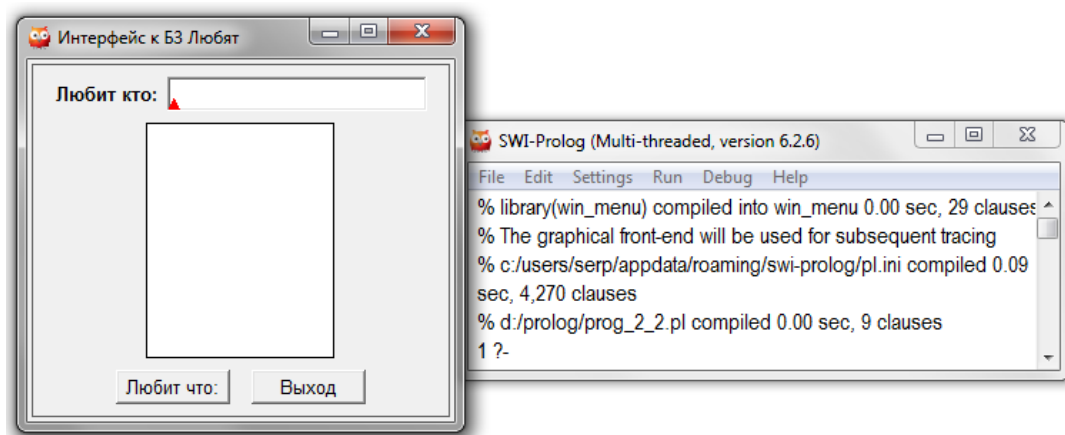


Рис. 3.23. Результат запуска ярлыка start_win_2.

Если считать, что экранная форма является единственно доступным интерфейсом к разработанной программе, то странным является наличие на экране еще и консоли SWI-Prolog'a. Избавиться от нее можно также за счет небольших манипуляций в строке запуска ярлыка start_win_2. С этой целью достаточно в стартовую цель добавить еще один предикат

```
|| -g gui_to_likes, win_window_pos([show(false)])
```

Запуск измененного ярлыка приведет к тому, что на экране появиться только одна экранная форма, а окно консоли будет скрыто. Именно это действие и выполняет второй предикат стартовой цели. Он входит в группу предикатов настройки и управления основным окном SWI-Prolog'a. Вместе с ним в этой группе win_has_menu, win_insert_menu_item, win_insert_menu, window_title и, естественно, win_window_pos. При этом аргумент предиката win_window_pos(+ListOfOptions) – это список структур, которые являются предопределенными и могут иметь один из стандартных видов: size(W, H), position(X, Y), zorder(ZOrder), activate и show(Bool).

Проведем еще один эксперимент по формированию строки запуска SWI-Prolog'a, но другого режима его работы. С этой целью надо выполнить следующую последовательность действий:

- скопировать ярлык start_win_1 в ярлык start_con_1,
- в строке запуска заменить swipl-win.exe на swipl.exe,
- имя файла goal_1.pl записать в полной спецификации, например, d:\prolog\goal_1.pl,
- сохранить ярлык в любой папке,
- обратить внимание на различие в отображении ярлыков start_win_1 и start_con_1 в проводнике Windows (рис. 3.21).

Если теперь щелкнуть мышкой по этому ярлыку, то на экране появится текстовая консоль Windows с результатами работы программы (рис. 3.24). Причем эти результаты будут точно соответствовать результатам, которые были получены ранее (рис. 3.22).

Главное их отличие заключается только в том, что в одном случае используется текстовая, а в другом графическая консоль SWI-Prolog'a. Но самым главным отличием является то, что для запуска одной и той же пользовательской программы использовались разные версии приложений из поставки SWI-Prolog'a.

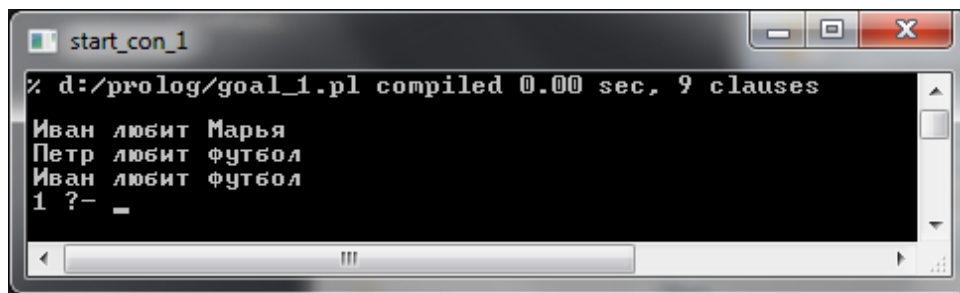


Рис. 3.24. Результат запуска ярлыка start_con_1.

Следует отметить, что в текстовой консоли, как и в графической, в ответ на приглашение «?–» можно вводить любые запросы, подгружать предикатом consult() новые программные файлы и т.д. Нет только среды для редактирования файлов программ, но для этого можно использовать любой текстовый редактор, например, Блокнот. Наличие в последней строке текстовой консоли приглашения «?–» свидетельствует о том, что мы находимся в среде Prolog'a и для выхода из него надо ввести halt.

Если описанный выше эксперимент повторить для start_win_2, то он закончиться неудачей, так как swipl.exe, являясь консольным приложением, не поддерживает работу с графическими средствами ХРСЕ. Но если при использовании текстовой консоли есть необходимость в интерактивных запросах, то решить этот вопрос можно путем некоторой модификации файла goal_1.pl до файла goal_2.pl

```
/* Программа goal_2.pl */
:- include('D:/Prolog/prog_1.pl').
goal :- write('Введи имя:'), read(A), !,
```

```
likes(A,S),write(A),
write(' любит '),write(S),nl,fail.
```

Не будем сейчас останавливаться на сути этой программы. Она станет ясна при более глубоком знакомстве с языком Prolog. Сейчас же, используя этот программный файл, на базе ярлыка start_con_1 создадим новый ярлык start_con_2. Единственным отличием в нем будет вызов программного файла godl_2.pl.

Если теперь запустить его, щелкнув по нему мышкой, то на экране появится текстовая консоль, в которой будет запрос имени. После ввода, который должен заканчиваться точкой, и нажатия Enter, окно примет вид, аналогичный тому, что приведен на рис. 3.25.

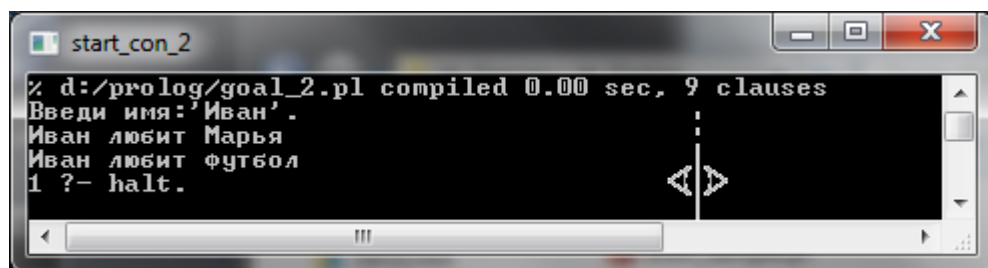


Рис. 3.25. Результат запуска ярлыка start_con_2.

Упражнение 3.6.

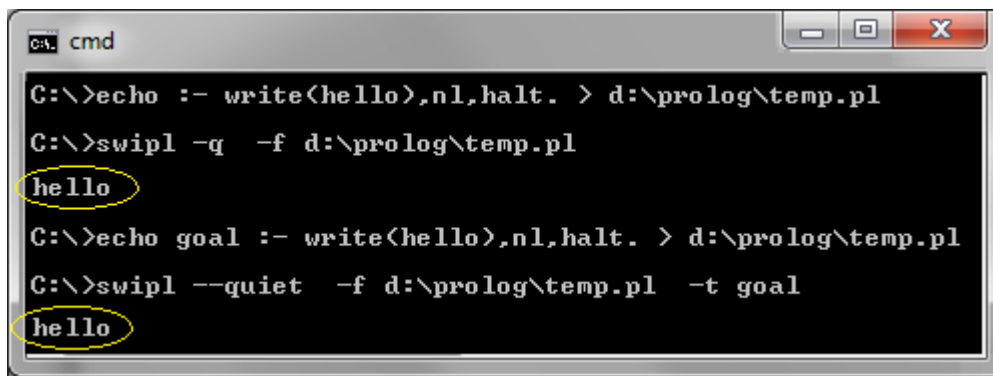


1. Создать ярлыки start_win_1, start_win_2, start_con_1, start_con_2 и исследовать их работу по запуску Prolog программ.
2. Открыть текстовую консоль Windows (командную строку) и в ней ввести команды, аналогичные командам запуска в ярлыках.

3.6.2. Использование командных и скриптовых сценариев

В самом начале этого параграфа было отмечено, что bin\swipl.exe – это консольная версия, которая используется в сценариях. В том числе и в командных сценариях, которые представляют собой последовательность команд, выполняемых непосредственно под управлением операционной системы. С этой целью создаются специальные командные файлы или файлы сценариев. Чтобы познакомиться с возможностью использования в этих файлах swipl.exe выполним сначала некоторую последовательность действий с использованием swipl.exe непосредственно в текстовой консоли Windows. Рассмотрим пример, который в командной строке (рис. 3.26):

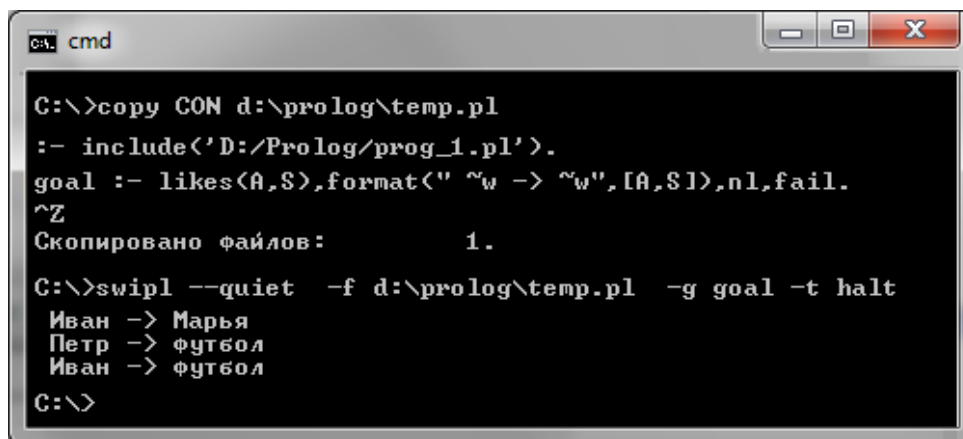
- Создает, с помощью команды echo, временный программный файл temp.pl, состоящий всего из одной Prolog строки.
- Запускает этот файл на выполнение с использованием консольного приложения SWI-Prolog'a.



```
C:\>echo :- write(hello),nl,halt. > d:\prolog\temp.pl
C:\>swipl -q -f d:\prolog\temp.pl
hello
C:\>echo goal :- write(hello),nl,halt. > d:\prolog\temp.pl
C:\>swipl --quiet -f d:\prolog\temp.pl -t goal
hello
```

Рис. 3.26. Создание файла Prolog-программы и ее выполнение.

Другим чуть более сложным является пример, который также прямо из консоли Windows позволяет создать временный файл Prolog-программы, которая является запросом к нашей тестовой базе знаний и запускает его на выполнение для форматированного вывода результатов.(рис. 3.27).



```
C:\>copy CON d:\prolog\temp.pl
:- include('D:/Prolog/prog_1.pl').
goal :- likes(A,S),format(" ~w -> ~w",[A,S]),nl,fail.
^Z
Скопировано файлов:          1.
C:\>swipl --quiet -f d:\prolog\temp.pl -g goal -t halt
Иван -> Марья
Петр -> футбол
Иван -> футбол
C:\>
```

Рис. 3.27. Создание файла Prolog-программы и ее выполнение.

Файл Prolog-запроса создается с помощью команды `copy CON <имя файла>`, которая позволяет вводить с клавиатуры несколько строк, заканчивая набор нажатием клавиш `Ctrl+Z`.

```
:- include('D:/Prolog/prog_1.pl').
goal :- likes(A,S),format(" ~w -> ~w",[A,S]),nl,fail.
```

Набранные на клавиатуре строки сохраняются в файле `temp.pl`, который затем выполняется `swipl.exe` и результаты выводит в эту же консоль. Это происходит потому, что в программах на Prolog'е используются предикаты `write/1` и `format/2`, которые выполняют вывод в стандартный поток.

Но при использовании командных и скриптовых сценариев появляется возможность этот поток перехватывать и перенаправлять к другим программам. Для обработки простых командных сценариев используют интерпретатор команд `Cmd.exe`, для более сложных – сервер сценариев

Windows CScript.exe. Последний понимает такие скриптовые языки, как JScript и VBScript..

Рассмотрим еще один пример, который результаты работы SWI-Prolog'a перенаправляет во входной поток программы на VBScript, которая в свою очередь принимает этот поток, обрабатывает его внутри своей программы и отобразит на экране. Упрощенно эту задачу можно решить всего одной строкой вида:

```
swipl.exe -f goal_1.pl -g goal -t halt | CScript read.vbs
```

Естественно, это возможно только при условии, что существует файл read.vbs, который имеет хотя бы такую простую структуру, как

```
' Программа read.vbs на VBScript
s = WScript.StdIn.ReadAll() ' Читаем входной поток
MsgBox s,, "Данные Пролога:
```

Однако в реальных условиях выполнения этого примера на конкретном компьютере какие-либо переменные окружения и пути доступа могут быть не прописаны. В этих условиях целесообразнее создать файл командного сценария, используя для этого обычный текстовый редактор.

```
REM Командный сценарий pl -> vbs (pl_vbs.cmd)
echo off
SET swi="C:\Program Files (x86)\swipl\bin\swipl.exe"
SET fvb= d:\prolog\read.vbs
SET fpl= d:\prolog\goal_1.pl
%swi% -f %fpl% -g goal -t halt | CScript %fvb%
```

После того, как содержимое файла будет полностью введено, необходимо сохранить этот файл, например под именем pl_vbs.cmd (или start.cmd). Обратите внимание на то, как будет выглядеть пиктограмма этого файла в проводнике Windows (рис. 3.22). До запуска файла командного сценария на выполнение следует проверить, что вы не забыли про файл read.vbs, что он вообще существует и находится в том месте диска, как указано в командном сценарии.

Запустив файл командного сценария на выполнение, на экране дисплея появится два окна (рис. 3.28). Одно из них будет представлять текстовая консоль, а второе - экранная форма, которая построена программой на VBScript'е. Следует отметить, что содержимое экранной формы будет точно соответствовать данным, которые были получены в результате работы программы goal.pl, как результат сформированного ею запроса к базе знаний prog_1.pl.

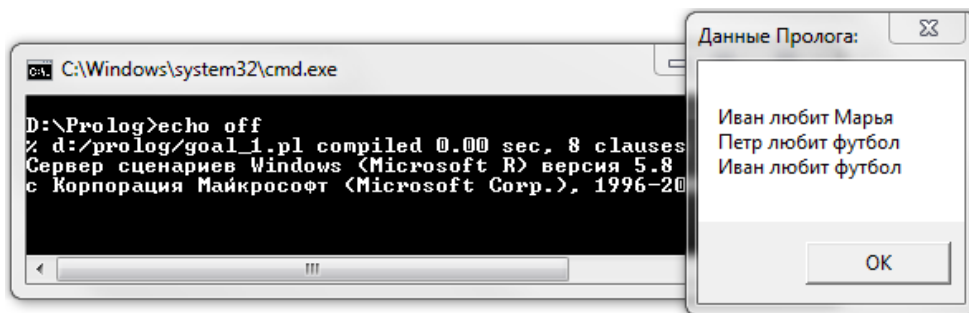


Рис. 3.28. Результат работы командного сценария.

Если обратить внимание на текстовую консоль Windows, то из анализа представленных там сообщений видно, что в выполнение данного командного сценария принимают участие SWI-Prolog и сервер сценариев Windows. Если на форме нажать кнопку OK, то это приведет к закрытию обоих окон.

Упражнение 3.7.

1. Сформировать файлы командных сценариев для первых трех примеров данного параграфа. Оттестировать их работу.
2. Подготовить все необходимые файлы для работы с предложенным командным сценарием. Отредактировать его под свои файлы.
3. Исследовать все взаимодействия, которые выполняются при взаимодействии SWI-Prolog'a и VBScript'a.
4. Реализовать рассмотренные подходы для доступа к файлу примеров SWI-Prolog'a \demo\likes.pl.
5. Создайте файл командного сценария query.cmd:

```
echo off
SET swi="C:\Program Files (x86)\swipl\bin\swipl.exe"
SET demo="C:\Program Files (x86)\swipl\demo\likes.pl"
%swi% -q -s %demo% -t %1,"format(' ~w -> ~w',[X,Y]),nl,fail"
```
6. Разберитесь с его структурой. Выполните из окна командной строки ряд запросов, вида: query "likes(X,Y)", query "likes(X,chips)", query "indian(X)", query "indian(X),not(mid(X))".

В дополнение к изложенному, хотелось бы обратиться к еще одному варианту использования консольной версии SWI-Prolog'a в сценариях на базе VBScript. Особенность рассматриваемого ниже подхода, заключается в том, что как запросы к базе знаний, так и запуск swipl.exe для реализации этих запросов будет осуществляться непосредственно из скрипта.

Сервер сценариев Windows (Windows Script Host), под управлением которого выполняются сценарии, обладает способностью работать в "невидимом" режиме, когда при запуске скриптов никаких вопросов или системных сообщений выводиться не будет. При использовании метода Exec(<Command>) объекта WSH будет создан новый дочерний процесс, который запускает заданное консольное приложение. Возвращаемый объ-

ект WshExec позволяет контролировать ход выполнения приложения и обеспечивает доступ к потокам StdIn, StdOut и StdErr этого приложения. Примером реализации этого подхода для поставленной выше задачи может являться приведенный ниже фрагмент программы:

```
' Программа query_1.vbs - запрос к Прологу из VBScript
file_pl = "C:\Program Files (x86)\swipl\demo\likes.pl"
goal_pl = "likes(sam,Y), italian(Y),write(Y), nl, fail."
MsgBox GetSwiAnser(file_pl,goal_pl), , "Ответ Пролога:"

Function GetSwiAnser(file, goal)
    swi = """C:\Program Files (x86)\swipl\bin\swipl.exe"""
    cmd = swi & " -f """ & file & """ -t """ & goal + """
    Set WshShell = WScript.CreateObject("WScript.Shell")
    Set WshExec = WshShell.Exec(cmd)
    GetSwiAnser = WshExec.StdOut.ReadAll
    Set WshExec = Nothing
    Set WshShell = Nothing
end Function
```

Запуск этого сценария приведет к тому, что на экране появится окно, аналогичное приведенному справа на рис. 3.30. Недостаток в том, что цель жестко прописана в теле сценария (переменная goal_pl). Следует дать возможность генерировать цель программно или хотя бы вводить ее интерактивно. С этой целью можно сделать минимальные правки в исходном сценарии, изменив в нем всего две строки:

```
' Программа query_2.vbs - запрос к Прологу из VBScript
file_pl = "C:\Program Files (x86)\swipl\demo\likes.pl"
goal_pl = InputBox ("Введите цель:", "Запрос к Прологу")
If IsEmpty(goal_pl) Then WScript.Quit 0
. . .
```

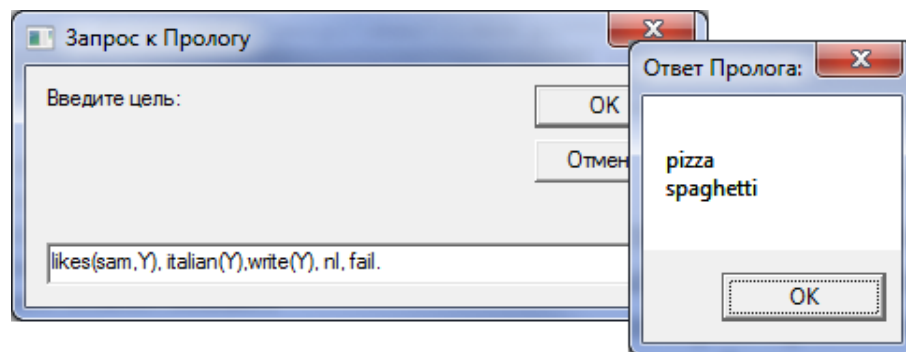


Рис. 3.29. Результат работы сценария query_2.vbs.

После запуска сценария и ввода цели экран будет аналогичен рис. 3.29.

4. СРЕДА РАЗРАБОТКИ TURBO-PROLOG

Наиболее мощной современной реализацией является Visual Prolog, в который по сравнению со стандартными реализациями добавлены средства объектно-ориентированного программирования, анонимные предикаты (лямбда-предикаты), императивные конструкции (foreach, if...then...else), коллекторы списков и многое другое. Visual Prolog является наследником системы Turbo Prolog, которая в конце 80-х и начале 90-х годов прошлого века являлась наиболее популярной, особенно в нашей стране, системой логического программирования. Turbo Prolog отличается от классического тем, что основан на строгой статической типизации. В отличие от других аналогов, он является компилируемым языком и отличается высокой скоростью трансляции и выполнения, что как раз и достигается за счет отхода в этой среде от классической реализации языка Prolog.

К настоящему времени накоплен большой объем как технической, так и учебной информации, посвященной реализации Prolog-программ именно в этой среде. Кроме того, исходные коды программ, написанных на Turbo Prolog'е могут импортироваться, загружаться и успешно работать также и в среде Visual Prolog.

Тысячи примеров программ из различных областей человеческой деятельности, реализованных средствами Turbo Prolog'а можно найти на просторах Интернет. В этих условиях среда Turbo Prolog'а представляет интерес и в современных условиях. Особенно в учебных целях, как первая ступенька в изучении строго типизированного языка до перехода к его объектно-ориентированным расширениям. В некоторых университетах и сегодня читаются развернутые курсы по Turbo Prolog'у.

Поэтому краткий экскурс в эту среду, по мнению автора, будет совсем не лишним. Единственное ограничение – это среда была реализована под DOS и если в старых версиях Windows спокойно запускалась как DOS приложение, то в последних версиях Windows, особенно 64-разрядных, она может не работать. В этом случае запуск возможен из среды виртуальной машины с Windows XP или среды эмулятора DOSBox (см. Приложение 1).

4.1. Структура программы на Turbo Prolog'e

Программа на Turbo Prolog'e состоит из нескольких секций, каждая из которых идентифицируется ключевым словом и имеет следующую обобщенную структуру:

```
domains
    /* секция объявления доменов */
database
    /* секция объявления динамических баз данных */
predicates
    /* секция объявления предикатов */
goal
    /* внутренняя цель -.подцель1, подцель2, ... */
clauses
    /* предложения (факты и правила) */
```

Обязательным в программе является наличие двух секций с именами `predicates` и `clauses`. В первой из них описываются структуры используемых в программе отношений, а во второй эти отношения определяются, то есть декларируются.

Turbo Prolog может иметь внутреннюю цель. В этом случае программа начинает выполняться с точкой входа в секции `goal`. Если внутренняя цель не задана, то в этом случае система работает в интерактивном режиме и пользователь может формировать запросы к программе в окне Диалога, аналогично тому, как было рассмотрено выше для других версий Prolog. Программа, которая в качестве примера была рассмотрена в предыдущих разделах, в среде Turbo Prolog будет выглядеть практически аналогично, но с добавлением секции описания типов используемых данных.

```
/* программа 3.1 */
predicates
    likes(string,string)          /* описание предиката */
clauses
    likes("Иван","Марья").        /* факт */
    likes('Петр', "футбол").      /* факт */
    likes("Иван",X) if likes("Петр",X). /*правило*/
```

В этой программе предикат `likes/2` описывает структуру отношения, домены которого имеют тип строки символов. Факты и правила записаны в виде предложений, каждое из которых заканчивается точкой. Текст, заключенный в `/* ... */` – это комментарий. Отсутствие в программе секции `goal` предполагает, что запросы будут осуществляться из оболочки Turbo Prolog. Запросы к программе формируются аналогично рассмотренному в предыдущих разделах. Например,

```
Goal: likes('Петр', X) .
X = футбол.
1
```

Однако использование в программах только секций *predicates* и *clauses*, является достаточным при разработке только простых программ, которые не используют описание данных и их структур, а также не работают с динамическими базами данных. Рассмотрим более подробно назначение каждой из секций программы.

Секция *domains*

В этой секции объявляются любые нестандартные домены, которые используются в качестве аргументов предикатов. Домены в Turbo Prolog являются аналогами типов в других языках. Основными стандартными доменами Turbo Prolog'a являются :

- *char* – символ, заключенный в одиночные кавычки (например, 'a');
- *integer* – целое от -32768 до 32767 (переводится в вещественное автоматически, если необходимо);
- *real* – вещественное (например, -68.72, 6e-94, -791e+21);
- *string* - последовательность символов, заключенных в двойные кавычки (например, "нажмите ввод");
- *symbol* – набор латинских букв, цифр и символов подчеркивания, где первый символ строчная буква (например, n_fax), либо заключенная в кавычки последовательность символов, которая начинается с большой буквы или имеет пробелы (например, "N fax ", "футбол").

Кроме стандартных доменов можно использовать свои, определяемые пользователем, домены. Для этого в области объявления доменов могут быть использованы следующие форматы:

- *name* = *stanDom*, где *stanDom* – один из стандартных доменов: *int*, *char*, *real*, *string* или *symbol*, а *name* - одно или несколько имен доменов. Пример: *year*, *height* = *integer* или *fio* = *symbol*.

- *mylist* = *elementDom**, где *mylist* – область, состоящая из списков элементов из области *elementDom*, которая может быть определена пользователем или иметь стандартный тип. Пример, *numberlist* = *integer** или *letter* = *char** ;

- *myCompDom* = *fun*₁(*d*₁₁,...,*d*_{1n}); *fun*₂(*d*₂₁,..., *d*_{2s}) ; ... *fun*_M(*d*_{m1},...,*d*_{mq}) , где *myCompDom* – область, которая состоит из составных объектов, описываемых указанием функтора и областей для всех компонентов, а *fun*₁ , *fun*₂, ... , *fun*_M – имена функторов. Правая часть этого описания может определять несколько альтернатив, разделенных символами «;» или «or».

Каждая альтернатива должна содержать единственный функтор и описание типов для компонентов d_{ij} . Например,

```
avto = car(symbol,integer) ,  
packing = box(integer,integer,integer) ; bottle(integer)
```

описывает две области `avto` и `packing`. Область `avto` соответствует двухкомпонентной структуре с функтором `car`, а область `packing` соответствует одной из двух возможных структур `box` и `bottle`, которые различаются не только именами, но и количеством компонент.

- `file = name1; name2; ... nameN`, где `file` – файловая переменная, а `name` принимает значение либо `keyboard`, `screen`, `printer`, `com`; `stdin`; `stdout`, либо другое, присвоенное пользователем и начинающееся со строчной буквы.

Рассмотрим пример. Пусть требуется разработать структуру данных для хранения информации о компьютерах. При этом каждый компьютер будем рассматривать как набор входящих в него устройств, среди которых могут быть: процессоры с указанием их наименования и частоты, НЖМД с указанием объема и фирмы изготовителя, а также мониторы определенного типа. Тогда область определения доменов может иметь вид:

```
domains  
  name , firm , type = symbol  
  freq , vol = integer  
  device = processor(name,f);disk(firm,vol);monitor(type)  
  computer = device*
```

В этом описании домен `computer` – это список элементов типа `device` и каждый элемент этого списка может иметь структуру типа либо `processor`, либо `disk`, либо `monitor`, содержащих одну или две компоненты, каждая из которых имеет стандартный символьный или целый тип.

Секция `predicates`

В секции `predicates` объявляются предикаты и типы аргументов этих предикатов. Имена предикатов должны начинаться со строчной латинской буквы, за которой следует последовательность букв, цифр и символов подчеркивания (до 250 знаков). В именах предикатов нельзя использовать символы пробела, минуса, звездочки, обратной (и прямой) черты. Объявление предикатов имеет форму:

```
predicates  
  predicateName1 (domen11, domen12, ..., domen1m)  
  . . .  
  predicateNamen (domenn1, domenn2, ..., domennk)
```

Здесь `domeni` – либо стандартные домены, либо домены, объявленные в секции `domains`. Объявление домена аргумента и описание типа аргумента - суть одно и то же. Количество аргументов предиката определяют его аргументность или размерность. Предикат может указываться только именем и не иметь аргументов. Обычно, имя предиката выбирается так, чтобы оно отражало определенный вид взаимосвязи между аргументами этого предиката. Пример описания предикатов:

```
predicates
    student(string, real)
    start
    good_student(string)
```

Можно использовать несколько описаний одного и того же предиката. При этом все описания должны следовать одно за другим и должны иметь одно и то же число аргументов. Пусть требуется определить отношение между тремя аргументами, первые два из которых соответствуют слагаемым, а третий - сумме двух первых. Этот предикат может быть описан в следующем виде

```
predicates
    add(integer, integer, integer)
    add(real, real, real)
```

и позволит его аргументам принимать значения как из области целых, так и из области действительных чисел.

Секция `clauses`

В секции `clauses` размещаются факты и правила, с которыми будет работать Turbo Prolog, пытаясь разрешить цель программы. Факт – это утверждение о существовании некоторого отношения между аргументами, обозначаемого именем предиката. Представляют собой фразы без условий, содержат утверждения, которые всегда абсолютно верны. В отличие от фактов, правила содержат утверждения, истинность которых зависит от некоторых условий, образующих тело правила. Факты и правила могут быть записаны в несколько строк, но обязательно должны заканчиваться точкой. В общем случае формат секции `clauses` имеет вид:

```
clauses
    /* факты *.
    predicateName_1(term_11, term_12, ..., term_1k).
    .
    .
    predicateName_N(term_N1, term_N2, ..., term_Ns).
```

```

/* правила */
goal_1 :- subGoal_11, subGoal_12, ... , subGoal_1k.
        . . .
goal_M :- subGoal_M1, subGoal_M2, ... , subGoal_1Mn,

```

где:

- predicateName_K – имена предикатов, которые до этого уже были описаны в секции predicates,
- term_KJ – аргументы предикатов (термы), количество которых должно соответствовать arity описания предиката,
- goal_N – заголовок правила, который имеет ту же форму, что и факт,
- subGoal_N1, ... , subGoal_Nk – тело правила, которое представляет собой список подцелей, разделенных запятыми (логическое И) или точкой с запятой (логическое ИЛИ).

Пример фактов, определяющих отношение, заданное описанным в секции predicates предикатом student, может иметь вид:

```

clauses
    student("Петров",4.5) .
    student("Сидоров",3.75) .

```

Пример записи правила

```

good_student( Name ) :- student( Name , B ) , B > 4.

```

Для того, чтобы заголовок правила был истинным, надо чтобы каждая подцель, входящая в тело правила, была истинной. Переменные заголовка квантифицированы универсально, а переменные, входящие только в тело правила, квантифицированы экзистенциально.

Секция goal

В секции goal задается внутренняя цель программы, что позволяет программе запускаться независимо от среды разработки. Если внутренняя цель включена в программу, то Turbo Prolog выполняет поиск только одного первого решения, и связываемые с переменными значения не выводятся на экран. Если внутренняя цель не используется, то в процессе работы есть возможность вводить в диалоговом окне внешнюю цель. При использовании внешней цели Turbo Prolog ищет все решения и выводит на экран все значения, связываемые с переменными.

В систему Turbo Prolog включено более 200 встроенных стандартных предикатов и более дюжины стандартных доменов. В случае использования этих предикатов и доменов нет необходимости объявлять их в программе.

Рассмотрим пример программы, в которой задана внутренняя цель и используется обращение к стандартным предикатам:

```
/* Программа 3.2 */
predicates
    hello
goal
    hello.
clauses
    hello :-
        makewindow(1,31,31,"My first programm",4,54,10,22),
        nl,write(" Please, type your name "),
        cursor(4,5),
        readln(Name),nl,
        write(" Wellcome ",Name).
```

В этой программе на экране формируется окно заданного размера и цвета, предикат `readln` запрашивает ввод с клавиатуры значения некоторой переменной (`Name`), после ввода которой предикат `write` выводит на экран приветствие. вместе со значением этой переменной. При этом предикат `makewindow(НомерОкна, ЦветЭкрана, ЦветРамки, Заголовок, Строка, Столбец, Высота, Ширина)` – определяет область экрана как окно, а предикат `cursor(Строка, Столбец)` – помещает курсор в указанную точку текущего окна. Перечень и назначение некоторых стандартных предикатов системы Turbo Prolog приведен в Приложении 2.

Однако чаще всего целью является сложный запрос к программе. Для разрешения какой-либо сложной цели Prolog должен разрешить все его подцели, создав при этом необходимое множество связанных переменных. Если же одна из подцелей ложна, то Prolog возвратится назад и просмотрит альтернативные решения предыдущих подцелей, а затем вновь пойдет вперед, но с другими значениями переменных. Этот процесс называется "поиск с возвратом".

Секция database

Ключевое слово `database` указывает на начало последовательности описаний предикатов динамической базы данных (ДБД). ДБД – это база, в которую факты добавляются во время исполнения программы, ее иногда еще называют внутренней или оперативной базой данных.

При разработке и работе программ часто возникает необходимость в хранении информации. В процедурных языках это реализуется посредством использования, например, глобальных переменных. В языке Prolog

(Turbo, PDC, Visual) подобное моделируется с помощью внутренней базы данных, которая и описывается в секции database.

Требования к описанию предикатов в этой секции, как и синтаксису их объявления, точно такие же, что и в секции predicates. Объявленные здесь предикаты не должны объявляться в секции predicates, но дизъюнкты этих предикатов могут присутствовать в секции clauses.

Факты, принадлежащие БД, обрабатываются отличным от обычных предикатов образом для того, чтобы ускорить работу с базой большого объема. Факты динамической базы могут модифицироваться в течение сеанса работы, добавляться и удаляться с помощью предикатов assert и retract, а также загружаться из файла с помощью стандартного предиката consult или записываться в файл предикатом save.

4.2. Оболочка системы Turbo Prolog

Для работы с системой достаточно запустить на выполнение файл gr.exe или prolog.exe, в зависимости от версии Turbo Prolog'a. На экране дисплея появится заставка с указанием текущей конфигурации системы на вашем компьютере.

Нажав на клавишу пробел, вы попадете в оболочку системы. При использовании русифицированной версии (gr.exe), ее вид будет аналогичен тому, что приведен на рис. 4.1, а при использовании версии Turbo Prolog 2.0 (prolog.exe) аналогичен тому, что приведен на рис. 4.2.

На экране отображается главное меню системы и четыре системных окна: редактирования, диалога, сообщений и трассировки. Эти окна могут быть использованы в любой конфигурации, и любое из них может занимать весь экран или его часть.

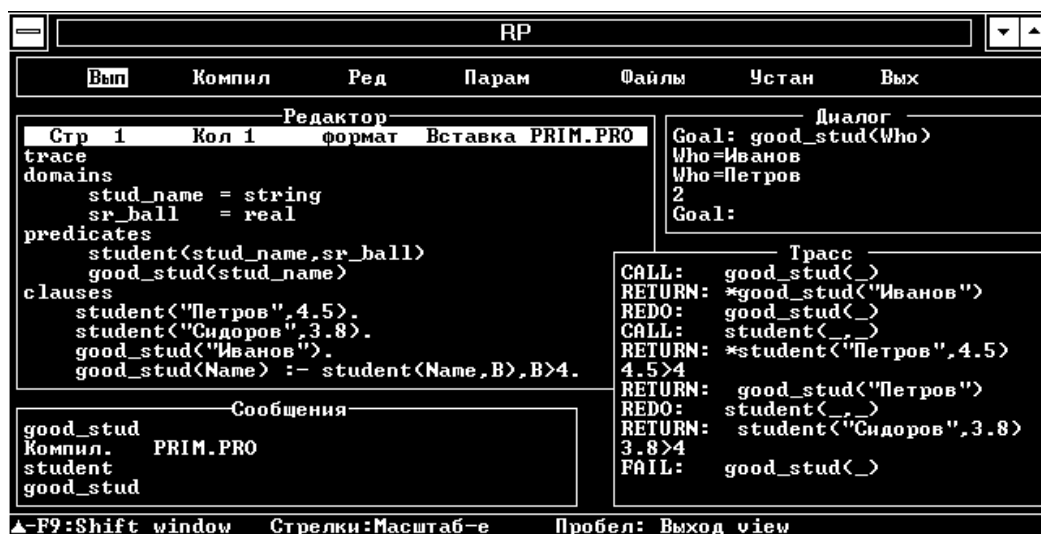


Рис. 4.1. Интерфейс русифицированной оболочки системы Turbo Prolog.

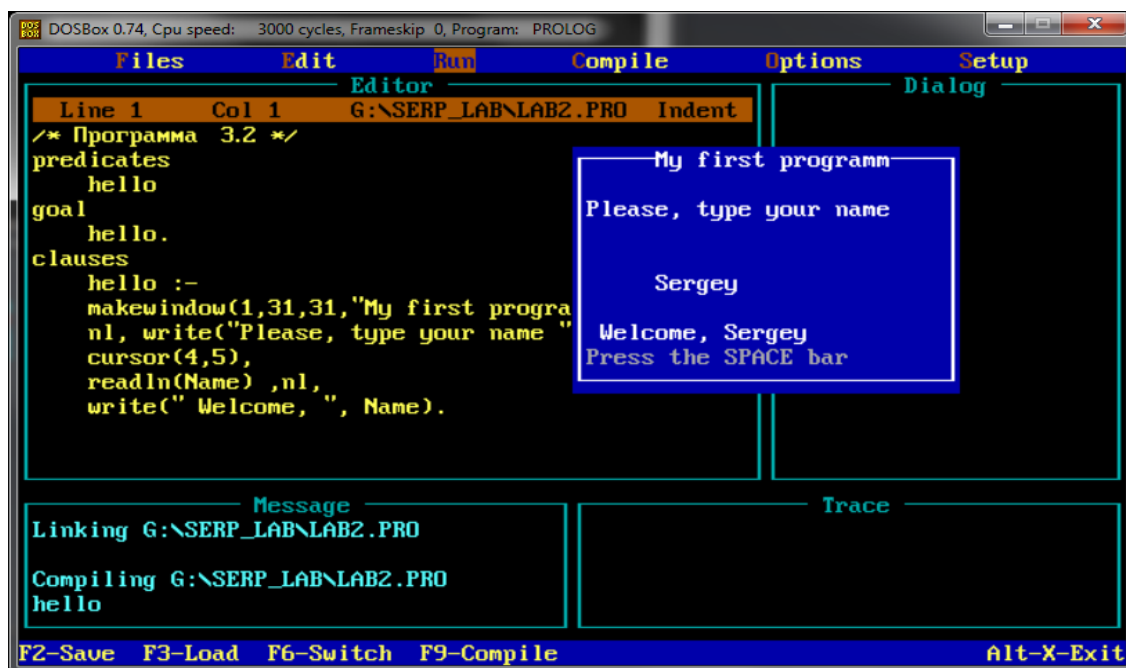


Рис. 4.2. Интерфейс Turbo Prolog 2.0 в эмуляторе DOSBox под MS Windows 7.

Нижняя строка экрана содержит сообщения о состоянии системы, описывая доступные команды и назначение функциональных клавиш. Назначение клавиш меняется при изменении режима работы. Главное меню содержит набор команд и подчиненных иерархических меню. Далее рассмотрим опции главного меню системы и дадим краткое описание их назначения для русифицированной версии. В системе Turbo Prolog 2.0 они практически такие же, но отличаются порядком следования в меню.

Опция меню "Редактировать" (Ред)

По этой команде вызывается встроенный текстовый редактор, который позволяет вводить и редактировать текст программы. Если не было указано имя программного файла, то по умолчанию оно – WORK.PRO. Методы работы с редактором такие же, как и с обычным текстовым редактором, а набор его команд близкий к стандартному набору команд для любых Турбо-систем. Перечень основных команд и комбинаций клавиш для вызова этих команд можно получить, нажав клавишу F1.

Особенностью встроенного редактора является наличие в его составе дополнительного окна, позволяющего одновременно работать с двумя файлами и обмениваться между ними блоками информации: копировать и переносить блоки программного кода из одного файла в другой.

Опция меню "Выполнить" (Вып)

Эта команда используется для выполнения откомпилированной программы, находящейся в памяти. При этом возможны две ситуации:

- Если цель содержится внутри программы, то есть в программе присутствует секция goal, то после выбора команды Вып результат работы программы будет представлен в окне Диалог. Нажатие на клавишу пробел вызовет возврат в главное меню.

- Если секция goal в программе отсутствует, то после выполнения команды Вып активизируется окно Диалог, в котором пользователь может вводить запросы в интерактивном режиме.

В ходе выполнения программы некоторые из функциональных клавиш имеют специальное назначение:

F8 – повторный ввод предыдущего запроса в окне Диалог;

F9 – вызов редактора;

Shift+F9 – выбор системного окна для изменения его размеров;

Shift+F10 – изменяет размеры или двигает окно;

Ctrl+F10 – окно на весь экран / стандартный размер.

Ctrl+S – остановка вывода на экран / продолжение вывода;

Ctrl+C или Ctrl+Break – прерывание исполнения программы.

Опция меню "Компилировать" (Компил)

По этой команде выполняется компилирование программы, которая загружена в окне редактора. Результат будет сохранен либо в памяти, либо на диске в виде *.OBJ или *.EXE файла, в зависимости от установки переключателя компиляции в меню "Режимы".

Опция меню "Файлы"

Выбор этой команды главного меню приводит к тому, что на экран дисплея выводится контекстное меню по работе с каталогами и файлами. Основные режимы этого меню по действию аналогичны этим режимам в других системах. Отметим только некоторые из них. Так, например, режим "Каталог" используется для выбора рабочего каталога. В частности каталог с именем PRO установлен по умолчанию, а для остальных следует указать путь к рабочему каталогу.

При выборе команды "Загрузить" система запрашивает имя файла. При этом можно ввести любое доступное для DOS имя файла. Если расширение в имени файла опущено, то система автоматически добавляет расширение *.pro. Если на сообщение системы "Имя файла:" будет нажата клавиша Enter, то отобразится содержимое текущего каталога и выбор файла надо выполнить клавишами управления курсором.

Режим "Переход к ДОС" вызывает временный выход в DOS, возврат из операционной системы в Turbo Prolog возможен после ввода команды exit.

Опция меню "Установки" (Устан)

Выбор этого режима вызывает на экран дисплея меню конфигурирования и установок оболочки системы программирования. В этом режиме можно изменять размер, позицию и цветовую палитру окон, устанавливать текущие каталоги для разных типов файлов (pro, obj, exe), сохранять или загружать текущую конфигурацию системы и т.д.

4.3. Отладка и трассировка программ

Любая программа после выбора в главном меню команды "Выполнить" запускается на компиляцию и выполнение. При этом система проверяет программу на соответствие синтаксису, отсутствие смещения значений из разных областей типов данных. Если при компиляции обнаруживается ошибка, то соответствующее сообщение появляется в нижней строке окна редактора и курсор в окне редактирования указывает на место ошибки.

Если программа успешно откомпилирована и система ожидает задание внешней цели для выполнения, то следующей стадией отладки является выбор таких запросов, чтобы программа тестировалась для достаточно широкого набора исходных данных. При возникновении непредвиденных ситуаций следует перейти на пошаговую трассировку программы. Для этого в текст программы вводится директива trace, которая может быть вставлена в программу перед секцией predicates.

При выполнении программы, содержащей директиву trace, в окне трассировки по очереди выводятся все цели и утверждения программы, используемые в процессе ее выполнения. Это позволяет следить за ходом выполнения программы, переходами от одной цели к другой, за процессом поиска с возвратом. Для пошагового перехода от выполнения одной цели к другой следует нажимать клавишу F10.

Директива trace показывает полную информацию, устраняя при этом различные оптимизации, выполняемые компилятором. Например, trace препятствует автоматическому уничтожению рекурсии, так, что можно наблюдать все возвраты предиката (RETURN). Пример трассировки программы для запроса good_student(Who) приведен на рис. 4.1. При выполнении пошаговой трассировки Turbo Prolog выдает следующие сообщения:

CALL: – вызов предиката в качестве текущей цели, дополнительно вводится имя предиката и значения его параметров.

RETURN: – это сообщение выводится, когда предложение выполнилось и предикат возвращает управление вызвавшему его предикату. Если существуют дальнейшие предложения, которые удовлетворяют входным параметрам, отобразится символ звездочка для указания того, что данное предложение находится в точке поиска с возвратом.

FAIL: – выводится в случае, если предикат не достиг успеха, за ним следует имя данного предиката.

REDO: - указывает на то, что имеет место поиск с возвратом. Имя предиката, который повторно выполняется, и значения его параметров выводятся за этим сообщением.

При отладке большой программы, содержащей множество предикатов, можно управлять ее трассировкой. Если, например, уже известно, что предикат `old_predicat(...)` работает успешно, то утверждение для предиката

```
new_predicat(...) :- old_predicat(...),  
                    new_goal(...), ...
```

может быть переопределено следующим образом:

```
new_predicat(...) :- trace(off), old_predicat(...), trace(on),  
                    new_goal(...), ...
```

и трассировка пойдет быстрее. Еще одной возможностью по управлению информацией, выводимой при трассировке, является использование директивы `trace` в расширенном формате

```
trace predicate1, predicate2, predicate3, ... ,
```

который выводит результаты обращений (`CALL`) и возвратов (`RETURN`) только для перечисленных в директиве `trace` предикатов.

4.4. Работа с простейшими программами в Turbo Prolog'e

Уже отмечалось, что любая программа на языке Prolog представляет собой текстовый файл, который для запуска из Turbo Prolog'a должен иметь расширение `*.pro`. Содержимое любого текстового файла или файла программы можно включать в другую программу на режиме компиляции. Для такой текстовой подстановки режима компиляции надо использовать директива компилятора `include`, которая имеет следующий синтаксис:

```
include "dos_file_name" ,
```

где `dos_file_name` - имя файла, включаемого в текущий программный файл, которое может включать и путь доступа к нему.

Включаемые файлы могут быть использованы только в естественных границах программы. Ключевое слово `include` может появиться только там, где допускается одно из ключевых слов `domains`, `predicates`, `clauses` или `goal`. Если в программе использована эта директива, то при компиляции в данное место текущей программы будет вставлено содержимое другого текстового файла, как части текущей программы. Это позволяет некоторый набор уже отлаженных описаний и предикатов хранить в отдельном файле

и включать его в другие программы при использовании в них обращений на эти, уже отлаженные, предикаты.

Включаемый файл сам может содержать директивы `include`. Однако, включаемые файлы не должны использоваться рекурсивно. То есть так, чтобы тот же самый файл включался более, чем однажды в процессе компиляции. Использование многих уровней включаемых файлов требует больше памяти в процессе компиляции, чем если бы те же самые файлы были включены непосредственно в главную программу.

Загрузка системы Turbo Prolog, ввод и запуск программ

До начала работы с Prolog-программами следует загрузить систему Turbo Prolog, войти в ее оболочку и, используя сведения, которые были изложены выше, выполнить следующие действия:

- ознакомиться с опциями главного меню и их назначением;
- установить режим компиляции программ в память компьютера;
- сконфигурировать, если это необходимо, размеры, положения и цветовые палитры всех окон;
- определить пути доступа к файлам, установив нужные для каждой категории файлов каталоги;
- выполненные установки записать в конфигурационный файл.

Первым этапом при работе с Prolog-программами является ввод их исходного текста, который выполняется в режиме редактирования. Для перехода в этот режим следует использовать опцию `Edit (Ред)` главного меню системы.

Упражнение 4.1.



1. Ввести в редакторе текст программы 3.1 из раздела 3.1. Закончив ввод, выйти из редактора в главное меню, нажав клавишу `Esc`.
2. Сохранить исходный код в файле с именем `prog_3_1.pro`.
3. Запустить его на выполнение, выбрав в главном меню `Run (Вып)`.

В данной программе нет секции `goal`, т.е. в программе отсутствует внутренняя цель, определяющая решение конкретной задачи. Такие программы могут использоваться только в среде системы Turbo Prolog. Поэтому, после ее запуска на выполнение системой активизируется окно `Dialog (Диалог)` и появляется приглашение на ввод внешней цели (`GOAL:`)

Работа с программой в режиме диалога

Внешние цели – это запросы к программе, которые пользователь может формировать в диалоговом окне. Введите запрос

```
GOAL: likes(Who, "футбол").
```

По окончании набора текста запроса нажать Enter. Проверьте, что в конце предложения стоит точка. До нажатия Enter запрос можно редактировать. Если в запросе нет ошибок, то в диалоговом окне Prolog выдаст сообщения:

```
Who=Петр
Who=Иван
2 Solutions
GOAL:
```

Упражнение 4.2.



1. Объясните, что обозначает данный запрос и какого он типа.
2. К каким элементам языка следует отнести такие объекты запроса, как "футбол", Who и likes.
3. Объясните полученный результат и смысловое значение выводимых в окне Dialog сообщений.

Система запоминает последний из введенных запросов. Для того, чтобы вызвать повторно предыдущий запрос, следует нажать функциональную клавишу F8. Вызовите повторно предыдущий запрос и отредактируйте его так, чтобы он имел вид

```
GOAL: likes (Who, "футбол", "Марья") .
```

Запустите его на выполнение. Аналогичные действия проделайте по вводу и запуску запроса вида:

```
GOAL: likes (Иван, X)
```

Упражнение 4.3.



1. Объясните полученные результаты, внесите изменения в запросы, чтобы они удовлетворяли синтаксису и семантике Prolog.
2. Повторно запустите запросы на выполнение.

Turbo Prolog, являясь строго типизированным языком, требует описания всех объектов. В связи с чем и предикат likes был описан как отношение между двумя символьными объектами, но какими из описания предиката, не ясно. Поэтому имеет смысл изменить описание этого предиката так, чтобы было ясно между какими объектами реального мира отношение likes устанавливается.

В частности, в данном примере, отношение likes определяется между некоторым лицом (person) и некоторым другим лицом или вещью (thing). То есть, это отношение соответствует конструкции русского языка вида:

“Person (подлежащее) likes (сказуемое) thing (дополнение)”

Для учета введенного дополнения, следует изменить в секции predicates описание предиката на новое

```
likes(person, thing)
```

Если запустить программу с этим исправлением на выполнение, то система выдаст сообщение об ошибке и в окне редактирования курсором отметить то место, где транслятор обнаружил ошибку. Текст сообщения об ошибке “Необъявленный домен или ошибка в написании” дает подсказку о том, что перейдя к использованию нестандартных (т.е. определенных пользователем) доменов мы забыли объявить их типы.

Так как областью изменения обоих, вновь определяемых, доменов являются символьные данные (точнее данные типа строки символов), то в программу должна быть добавлена секция `domains`, где должны быть объявлены нестандартные домены и их типы. Для данной программы она может иметь один из двух возможных видов:

```
domains                                domains
    person = string                    или    person, thing = string
    thing  = string
```

Введите эти добавления в программу, запустите ее на выполнение и задайте какие-либо запросы, из тех, что вводились ранее. Результат должен быть аналогичен предыдущему.

Трассировка программ в среде системы Turbo Prolog'a

Познакомиться с тем, как Prolog выполняет поиск ответов на запросы, а также отследить последовательность согласования фактов и правил можно при пошаговом режиме ее выполнения. Для перехода в режим пошагового выполнения программы (трассировки) надо в первой строке программы вставить директиву `trace`.

Эта директива при выполнении программы обеспечит трассировку всех предикатов. Запустим программу на выполнение и, помня о возможности вызова предыдущего запроса с помощью F8, зададим один из ранее выполнявшихся запросов. Например,

```
GOAL: likes(Who, "футбол").
```

Осуществим, с использованием клавиши F10, пошаговое выполнение программы, тщательно отслеживая при этом все перемещения курсора в окне редактора по тексту программы и регистрируя все выводимые в окне трассировки сообщения.

	Положение курсора в тексте программы
CALL: likes(, "футбол")	на факте1
REDO: likes(, "футбол")	на факте2
RETURN: *likes("Петр", "футбол")	на факте2
REDO: likes(, "футбол")	на заголовке правила

CALL:	likes("Петр", "футбол")	на теле правила
	. . .	на факте1
REDO:	likes("Петр", "футбол")	на факте2
RETURN:	likes("Петр", "футбол")	на факте2
RETURN:	likes("Иван", "футбол")	на теле правила

Провести анализ полученного результат будет значительно проще, если обратиться к материалу, изложенному ранее в разделе 2.2.



Упражнение 4.4.

1. Разберитесь в последовательности доказательства цели системой..
2. Выполните трассировку запросов likes(,"футбол") и likes(X,Y).

Работа с программами, содержащими внутреннюю цель

Для того, чтобы познакомиться с построением и работой программ, которые используют внутреннюю цель, модифицируем рассматриваемый пример таким образом, чтобы один из ранее использованных запросов, явился бы внутренней целью программы. Для этого в программу следует добавить еще одну секцию goal, в которой будет описана основная цель, решаемая программой. Пусть, например, эта цель будет аналогична запросу предыдущего параграфа. Тогда секция goal примет вид:

```
goal
    likes(Who, "футбол") .
```

Затем модифицированную программу запустим на выполнение. Если при запуске на решение у Вас появилось сообщение о синтаксической ошибке, при мигающем курсоре в соответствующем месте экрана, то посмотрите – не забыли ли Вы о том, что каждое предложение Prolog'a должно заканчиваться точкой.

Если в программе отсутствуют синтаксические ошибки, то после запуска ее на выполнение в окне диалога появится сообщение "Нажмите ПРОБЕЛ", которое свидетельствует о том, что программа отработала и запрос выполнен.

Но тогда возникает вопрос: "А где же результаты запроса?". Все дело в том, что в сформированной цели дается запрос о согласовании переменной Who с предложениями программы, а об отображении каких-либо данных или результатов, полученных в результате работы программы, ничего не говорится. В отличие от диалогового режима работы, при котором внешняя цель формулируется в виде запроса к программе, а система сама управляет процессом поиска и отображения результатов в некотором стандартном виде, при формировании внутренней цели все эти функции возлагаются на разработчика программы.

Формирование внутренней цели программы требует от пользователя не только задания запроса, но и обеспечения отображения его результатов на экране. Поэтому изменим исходную цель, добавив в нее еще одну подцель, которая будет обеспечивать отображение термина на экране дисплея:

```
goal
    likes(Who, "пиво"), write(Who).
```

Запустим на выполнение программу, в которой цель представляет собой уже конъюнкцию двух подцелей. В ходе выполнения этой программы в окне диалога выдается сообщение об одном из любителей футбола,

```
Петр
Нажмите ПРОБЕЛ
```

а после нажатия клавиши пробел, происходит остановка программы и выход в главное меню системы.

Связано это с тем, что при значении Who="Петр" каждая из подцелей принимает значение истина и вся цель становится истинной. Именно это и приводит к окончанию процесса вывода. Таким образом, мы обнаружили еще одно существенное отличие в формулировке одного и того же запроса в виде внешней или внутренней цели.

Внимание!!!



Если при задании внешней цели Turbo Prolog сам, управляя поиском, ищет все удовлетворяющие запрос ответы, то при использовании внутренней цели находится только первый из них.

Если цель решения задачи должна полностью соответствовать запросу и ее требуется включить в тело программы, а домены отношения, для большей определенности, должны быть поименованы, то в этом случае программа 3.1 может быть представлена в виде:

```
/* Программа 3.3 */
domains
    person, thing = string
predicates
    likes(person, thing)
goal
    likes(Who, "пиво"), write(Who), nl, fail. /* цель */
clauses
    likes("Иван", "Марья"). /* факт */
    likes("Петр", "пиво"). /* факт */
    likes("Иван", X) :- likes("Петр", X). /* правило */
```

В этой программе домены отношения likes имеют имена person и thing, которые имеют тип строки символов. Цель решения состоит из четырех подцелей, соединенных между собой запятыми, которая соответствует ло-

гической функции "И". То есть цель решения задачи представляется конъюнкцией подцелей. Цель будет достигнута и примет значение истина, если каждая из подцелей будет истинной. Подцели данной программы содержат: один определенный пользователем предикат likes и три встроенных стандартных предиката:

- write(Term) – выводит терм на дисплей,
- nl – обеспечивает переход на новую строку,
- fail – вызывает неудачу при доказательстве целевого утверждения.

Включение в основную цель дополнительных подцелей связано с тем, что задание цели внутри программы требует от пользователя не только формулирования запроса, но и обеспечения отображения его результатов на экране, а также обеспечение поиска всех удовлетворяющих запросу значений.

Так, если в основной цели будет отсутствовать четвертая подцель, то Prolog найдет только одного любителя футбола, а именно Петра. Поэтому включение в основную цель предиката fail вызывает состояние неудачи при доказательстве целевого утверждения и переход к повторному его доказательству при иных значениях.

Упражнение 4.4.



1. Отредактируйте свою программу до состояния программы 3.3, запустите на выполнение и проведите ее тестирование.
2. Если цель достигнута, то сохраните программу в рабочем каталоге на диске под именем LAB1_1.PRO.

Простейшая программа ввода-вывода данных

Программы, которые рассматривались до сего момента предполагали, что интерфейсом ввода-вывода является среда Turbo Prolog'a, а точнее его диалоговое окно. Но для большинства прикладных программ, работающих с базами данных, это недостаточно и нужен интерфейс, который более дружелюбен к пользователю.

С этой целью рассмотрим еще один пример программы с внутренним описанием цели, которая демонстрирует простейшие возможности Turbo Prolog'a по организации пользовательского интерфейса с использованием стандартных предикатов Турбо-Пролога.

Для этого надо загрузить программу 3.2 из раздела 3.1. Разобраться в ее структуре, познакомиться с синтаксисом, семантикой и назначением стандартных предикатов, используемых в данной программе. и запустить эту программу на выполнение. Результат должен быть аналогичен тому, что приведен на рис. 4.2.

Упражнение 4.5.



1. Установив режим трассировки, ознакомиться с ходом выполнения программы и действием стандартных предикатов.
2. Модифицировать программу таким образом, чтобы окно создавалось в середине экрана и в другой цветовой палитре.
3. Отладить программу и сохранить под именем LAB1_2.PRO

Компиляция программ.

Как уже неоднократно отмечалось, что особенность Turbo Prolog'a, как и PDC-Prolog, и Visual Prolog, заключается в том, что это компилируемые языки программирования, позволяющие получить загружаемые файлы, которые могут работать вне среды программирования.

Во всех предыдущих примерах Turbo Prolog до запуска программы на выполнение тоже выполнял компиляцию, но внутреннюю в своей памяти, не создавая при этом дополнительных файлов. Если же необходимо создать приложение, которое работало бы вне среды Turbo Prolog'a и запускалось бы прямо из операционной системы, то следует выполнить компиляцию в EXE file (рис. 4.3).

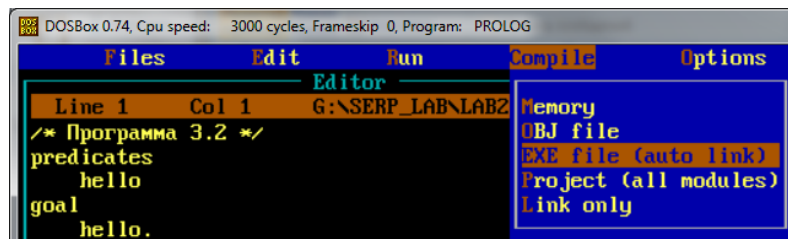


Рис. 4.3. Выбор режима компиляции в EXE file.

При этом в окне сообщений будет выведен полный протокол действий, выполненных Turbo Prolog'ом в процессе компиляции. Этот протокол будет заканчиваться вопросом о необходимости запуска на выполнение только что откомпилированной программы (рис. 4.4).

```
. . . .
Compilation successful
Linking <путь к файлу>\PROG_3_2.PRO
Execute (y/n):
```

При положительном ответе программа будет запущена из среды Turbo Prolog'a и результат ее работы будет аналогичен ранее полученному. При отрицательном ответе Turbo Prolog вернется в главное меню в режим компиляции. Но это не значит, что система ничего не сделала. Если теперь войти в рабочий директорию, то там можно найти три новых файла:

```
PROG_3_2.EXE, PROG_3_2.OBJ и PROG_3_2.SYM
```

Первый из них – это загрузочный файл, который получен в результате компиляции исходного кода. Он является законченным DOS приложением и может быть запущен под DOS или MS Windows XP непосредственно из операционной системы. В версиях MS Windows 7 и выше эту программу можно запустить из какой-либо среды, поддерживающей запуск DOS приложений (рис. 4.4).

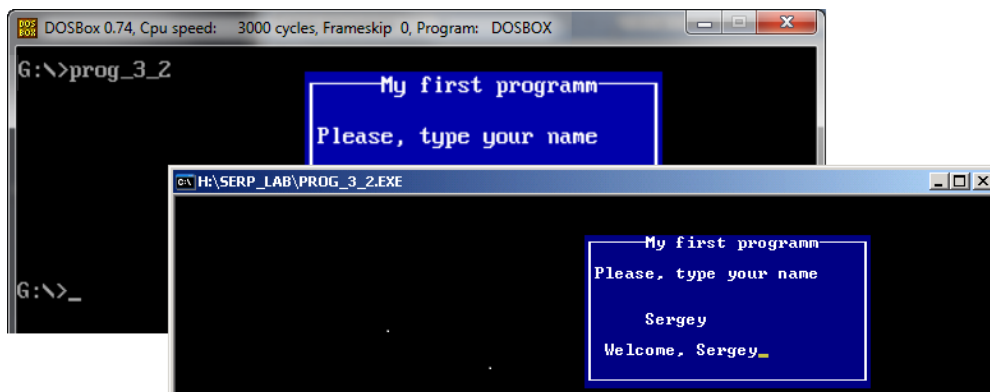


Рис. 4.4. Работа prog_3_2.exe в среде MS Windows XP и в среде эмулятора DOSBox под MSWindows 7.

Но, естественно, у вас проблем с запуском этой программы не должно возникать, так как вы работаете с системой Turbo Prolog, которая сама является DOS приложением. Поэтому она должна спокойно запускаться в той же среде, в которой запускается и сам Turbo Prolog.

Следует сделать небольшое замечание связанное с тем, что если при запуске этой программы в ответ на ввод имени и нажатия Enter окно программы сразу исчезает, то следует его несколько притормозить, чтобы была возможность прочитать выдаваемое программой приветствие. Этого можно добиться, если в описании предиката hello в последней строке добавить стандартный предикат readchar(_), который будет ожидать ввода с клавиатуры любого символа.

Упражнение 4.6.



1. Выполнить компиляцию исходной программы lab1_2.PRO.
2. Осуществить тестирования ее работы вне системы Turbo Prolog'a.
3. Используя директиву include, превратить lab1_2.pro в интерфейс для доступа к lab1_1.pro (рис. 4.5).
4. Отладить, сохранить под именем lab1_3.pro, затем откомпилировать.

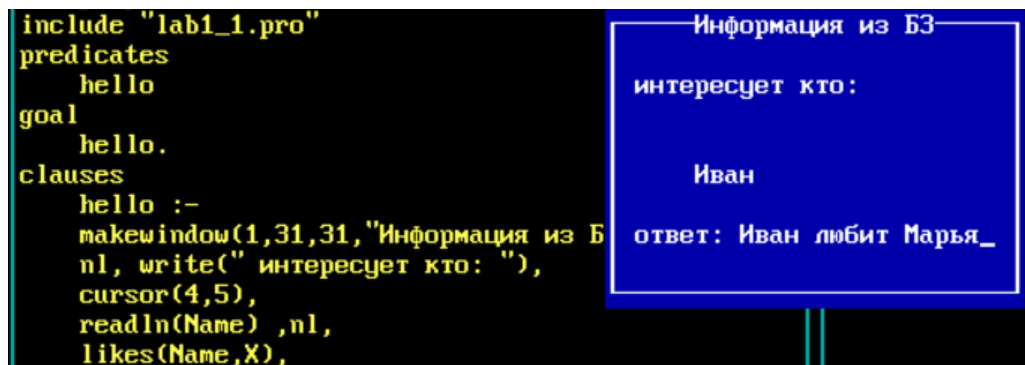


Рис. 4.5. Один из вариантов модифицированной программы.

4.5. Работа с графикой в Turbo Prolog'е

При решении ряда интеллектуальных задач таких, как семантический разбор предложений, определение подчиненности типовых узлов и блоков в каком-либо техническом изделии или взаимосвязь химических элементов в синтезируемом соединении удобно отображать в некоторой графической форме. Для этих целей в Turbo Prolog встроены ряд предикатов для работы с графикой.

Не вдаваясь в суть вопроса, а тем более в подробное изучение этих возможностей, рассмотрим небольшой пример. Единственная его цель заключается в том, чтобы только дать представление о наличии таких средств в системе Turbo Prolog'a.

```

/* Программа 3.3 */
predicates
    fig(integer)
goal
    graphics(5,0,31),
    fig(3), fig(1), fig(2),
    penup, left(180), forward(15000),
    pendown, fig(2),
    readchar(_).
clauses
% Треугольник
fig(1) :- pendown, forward(10000),
          left(120), forward(10000),
          left(120), forward(10000).
% Звезда
fig(2) :-
    forward(5000), right(144), forward(5000),
    right(144), forward(5000), right(144),
    forward(5000), right(144), forward(5000),
    right(144), forward(5000).
% Прямая

```

```
fig(3) :- line(10500,5000,19500,16000,2).
```

Для перехода в графический режим используется предикат `graphics(ПараметрРежима, Палитра, Фон)`, возврат в текстовый режим обеспечивает стандартный предикат `text`. Возможные значения параметров предиката `graphics` должны находиться в области предопределенных в Turbo Prolog'е констант, которые можно найти в справке по системе.

Вызов предиката `dot(Строка, Столбец, Цвет)` приводит к размещению точки в заданной позиции экрана. Значения Строка и Столбец – это целые числа от 0 до 31999. Аналогично предикат `line(Row1, Col1, Row2, Col2, Color)` определяет положение линии на экране.

Наряду с обычными графическими предикатами, в Turbo Prolog'е реализованы и возможности черепашьей графики, при которой в качестве начальной точки перемещения пера принимается центральная точка экрана. Действие предикатов черепашьей графики определяется в зависимости от следующих факторов:

- направление движения,
- рисует перо или нет,
- цвет пера.

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

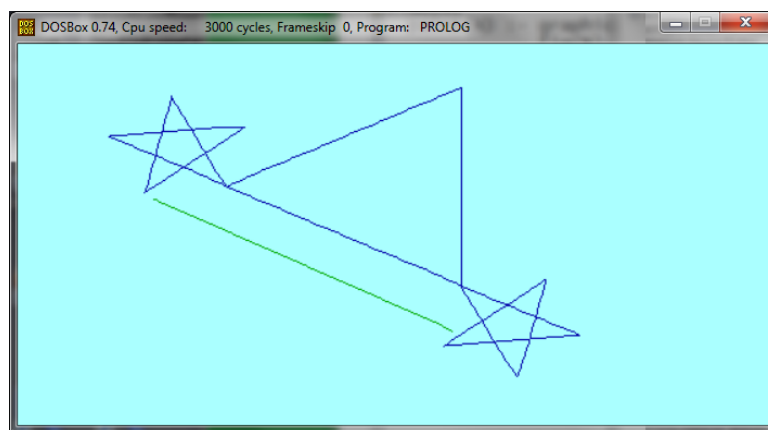


Рис. 4.6. Результат работы программы.

Движение пера управляется четырьмя стандартными предикатами: `forward` (вперед), `back` (назад), `right` (вправо) и `left` (влево). Например, предикат `forward(Step)` показывает, на сколько шагов должно переместиться перо. Чтобы повернуть перо вводится предикат `Angle` (Угол), где угол измеряется в градусах. Например, `right(144)` поворачивает перо вправо на 144 градуса, а `left(120)` – влево на 120 градусов.

Естественно, по нынешним временам, это не графика. Но сила базового пролога в механизме логического вывода, в возможности использования фактов и правил для описания баз знаний. И в этом плане возможности Turbo Prolog'a не очень-то уступают своим последователям. Тем более, что синтаксис и семантика многих конструкций и стандартных предикатов немного отличается от Turbo Prolog'a.

Но что касается возможностей по реализации готовых проектов, с развитым графическим интерфейсом, то здесь отставание налицо от своего сегодняшнего собрата, которым является Visual Prolog.

5. СРЕДА РАЗРАБОТКИ VISUAL PROLOG

На сегодняшний день Visual Prolog, несомненно, является одной из наиболее мощных сред реализаций Prolog'a. Будучи наследником системы Turbo Prolog, он является компилирующей средой и основан на строгой статической типизации данных. В этом его принципиальное отличие от стандартной реализации Prolog'a. Кроме этого в его составе присутствуют средства визуального и объектно-ориентированного программирования, императивные конструкции, коллекторы списков и многое другое. Общий объем описаний на систему составляет несколько сотен страниц. В этом разделе будут рассмотрены, и то очень кратко, только начальные сведения о простейших приемах работы в самой среде по разработке элементарных программных проектов. Причем, отойдя от изучения Prolog'a как языка декларативного программирования, основное внимание будет уделено методам работы в интерфейсе самой среды разработки Visual Prolog.



Замечание.

Более подробные начальные сведения по среде Visual Prolog можно получить, знакомясь с мануалами от разработчика, которые имеются на сайте http://wiki.visual-prolog.com/index.php?title=A_Beginners_Guide_to_Visual_Prolog.

5.1. Интегрированная среда разработки

Интегрированная среда разработки (Integrated Development Environment – IDE) используется для создания, разработки и поддержки проектов Visual Prolog. Ее используют на всех этапах жизни проекта для следующих задач:

- Создание – если проект создается средствами IDE, то в момент его создания выбираются основные свойства проекта, такие как: будет ли это исполняемое приложением или DLL, используется текстовое взаимодействие или пользовательский графический интерфейс и т.д.
- Построение – проект компилируется и связывается, используя IDE.
- Просмотр – компилятор и IDE собирают информацию о проекте, которая затем используется для быстрой навигации по проекту, и т.д.

- Разработка. – При разработке и поддержке проекта IDE используется для добавления и удаления в проект исходных файлов и элементов пользовательских интерфейсов, а также их редактирования.
- Отладка. – Отладчик IDE позволяет идти по тексту программы и наблюдать состояния ресурсов программы в ходе ее исполнения.

Знакомство с любой новой системой лучше всего начинать с простого и понятного примера. Разработаем в Visual Prolog с использованием его IDE проект для вычисления факториала некоторого числа. На языке базового Prolog'a вычислительная часть этой задачи может быть представлена двумя правилами на основе предиката `fact(<число>, <результат>)`:

```
fact(N, 1)      :- N<1, !.
fact(N, N*F1)   :- fact(N-1, F1).
```

Символ «!» в первом правиле обозначает отсечение и предотвращает попытку системы применить второе правило для случая $N = 0$.

5.2. Разработка консольного проекта

Среда Visual Prolog'a предоставляет возможность создания двух типов проектов: консольных, которые работают в командной строке Windows, и проектов с графическим интерфейсом, которые являются полноценными Windows приложениями. Знакомство с IDE начнем с создания консольного проекта.

Этап создания проекта

Любой тип проекта, который надо разработать, начинается с того, что он должен быть создан. Для этого надо выполнить следующие шаги:

- Запустить саму среду Visual Prolog'a.
- Выбрать в его главном меню опцию `Project -> New`. Откроется диалоговое окно `Project Settings`, в котором шесть вкладок (рис. 5.1).

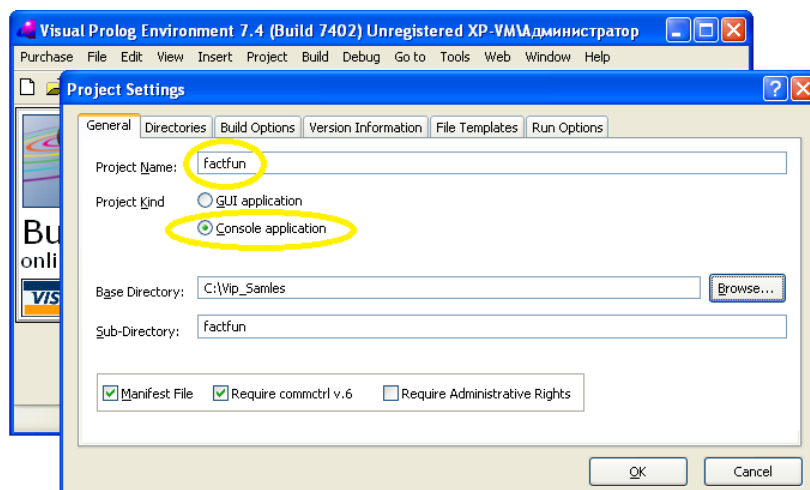


Рис. 5.1. Вид диалогового окна Project Settings.

- На первых этапах работы с IDE достаточно заполнить только вкладку General, где требуется указать имя проекта, выбрать его тип и указать место его хранения на диске.
- После заполнения нужных полей на вкладке, надо нажать кнопку ОК.

Система автоматически создаст проект, выполнит его компиляцию. При этом последовательность действий, выполняемых системой, выводится в диалоговом окне Messages. После окончания этого процесса среда Visual Prolog будет иметь вид, аналогичный тому, что приведен на рис. 5.2.

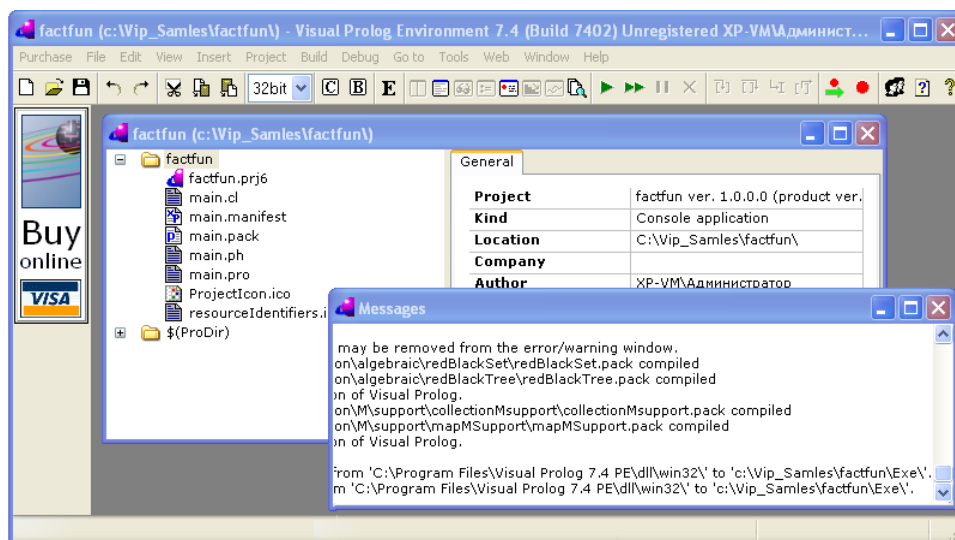


Рис. 5.2. Среда Visual Prolog по окончании создания проекта factfun.



Замечание.

Если автоматическая компиляция не выполнялась, что может быть в более ранних версиях системы, то выполните ее сами, выбрав опцию Build → Build, чтобы внести прототип класса factfun в дерево проекта.

Дерево проекта (Project Tree)

В процессе создания, сформировался прототип проекта – класс factfun, с именем, которое было дано проекту. Обратим внимание на дерево этого проекта (Project Tree), которое представлено в Project Window (рис. 5.3).

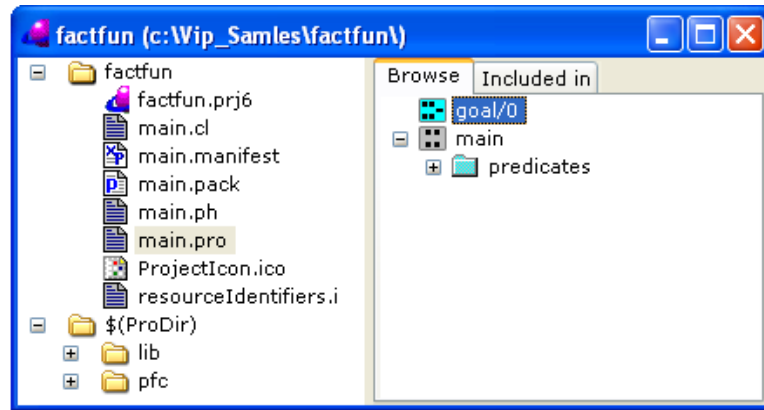


Рис. 5.3. Вид дерева проекта factfun.

Само по себе это дерево имеет стандартный для Windows вид, поэтому не будем останавливаться на способах его использования, а рассмотрим его содержание. Самый верхний узел представляет собой проект и проектную директорию. В самом низу расположен логический узел `$(ProDir)`, который представляет папку, в которой установлена сама система Visual Prolog. Эта папка содержит, как видно из рис. 5.3, библиотеки (`lib`) и исходные тексты классов системы Visual Prolog (`pfc`).

Упражнение 5.1.



Если проект создан, то используя Проводник Windows, убедитесь в создании прототипа проекта в указанной вами области диска (Base Directiry) и наличии там всех файлов, отображенных в дереве.

Из дерева проекта видно, что в состав проекта IDE включил ряд файлов, для которых в Visual Prolog имеет следующие специальные соглашения:

- `.ph` – заголовки пакетов (package headers), пакет – это набор классов и интерфейсов, которые используются совместно.
- `.pack` – содержат исполняемые разделы или конкретизации файлов, перечисленных в соответствующих `*.ph` файлах (packages).
- `.i` – содержит интерфейс (interface).
- `.cl` – содержит декларацию класса (class declaration)
- `.pro` – файл содержит имплементацию класса (class implementation).

Существует ряд стандартных действий, которые доступны при работе с деревом проектов. Правая кнопка мышки вызывает контекстное меню с операциями, которые можно выполнить над текущим узлом (рис. 5.4).

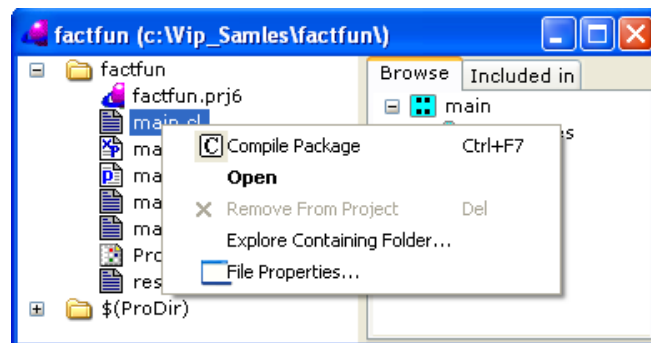


Рис. 5.4. Контекстное меню для узла main.cl.

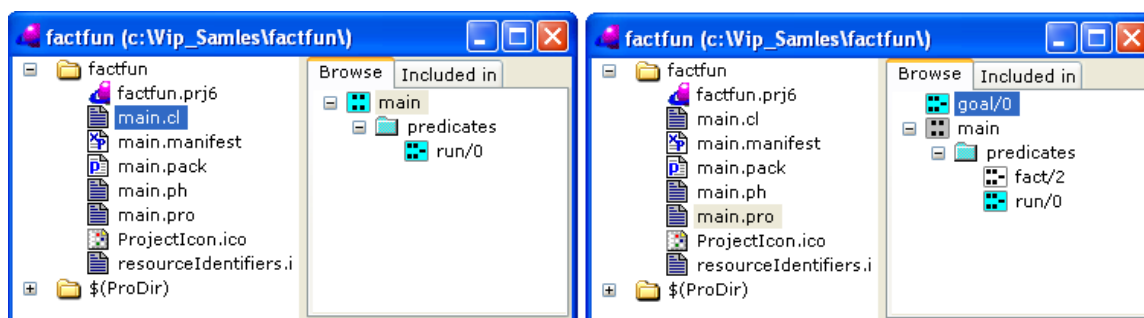


Рис. 5.5. Представление предиката run в main.cl и main.pro.

Некоторые сущности представлены в дереве дважды, поскольку они имеют как декларацию, так и имплементацию. Например, предикат run в классе main (рис. 5.5).

Двойной щелчок на узле вызывает редактор соответствующего элемента. Все коды редактируются с помощью текстового редактора, а ресурсы на основе окон, такие как диалоги, меню, редактируются с помощью графических редакторов. Графические редакторы мы увидим в следующем разделе, сейчас попытаемся вызвать текстовый редактор.

Этап разработки программного кода проекта

Если установить фокус на узел с именем класса main.pro, то в правой панели отобразятся все объекты, используемые и описанные в данном классе. При клике мышкой по обозначению этого класса, откроется окно со сгенерированным IDE прототипом программного файла проекта, который приведен слева на рис. 5.6.

Принимая во внимание, что implementation в переводе с английского — это реализация, приступим к реализации задачи вычисления факториала числа, используя при этом синтаксис и семантику Visual Prolog'a. Сейчас на этом вопросе останавливаться не будем, а отметим, что для реализации поставленной задачи файл main.pro должен иметь вид, аналогичный тому, что приведен справа на рис. 5.6 .

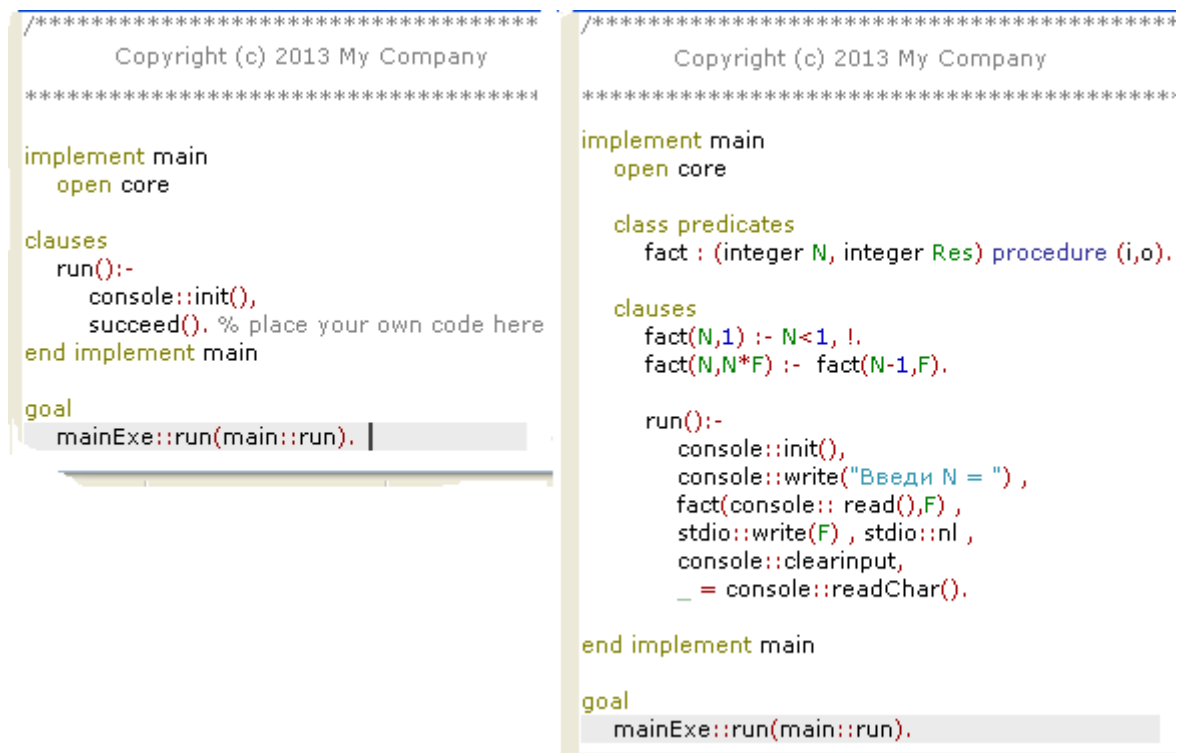


Рис. 5.6. Прототип программного файла проекта factfun и его реализация.

Следует отметить, что программа на Visual Prolog'е по своей сути очень напоминает Turbo Prolog: обязательное описание типов, те же секции, те же конструкции правил и фактов. Очень похожи и требования к синтаксису большинства конструкций. Даже имена некоторых стандартных предикатов и функций совпадают. Только в Visual Prolog'е они реализованы в виде методов предопределенных классов. Налицо объектно-ориентированная парадигма, присутствующая в современных версиях.

Следует отметить, что IDE Visual Prolog'е имеет достаточную развитую систему помощи (Help), контекстно-зависимую справку и подсказки при наборе текста. Полноценную документацию, как встроенную в систему, так и доступную в Интернете, можно получить, используя опцию Help из главного меню. Назначение любой конструкции можно узнать, если на нее переместить курсор и нажать клавишу F1. При наборе тех или иных строк, IDE, после ввода первых символов, выдает подсказки о возможных для ввода конструкциях (рис. 5.7).

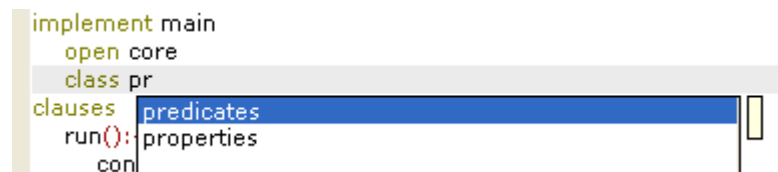


Рис. 5.7. Окно подсказок при наборе.

После того, как файл `main.pro` будет отредактирован, проект полностью готов к работе.

Этап компиляции и запуска проекта

Его можно запустить в работу, используя для этого в главном меню `Bild -> Run in Windows`. При выборе этой опции IDE сам выполнит компиляцию проекта, его загрузку и запуск на выполнение. Если не будет выявлено ошибок при компиляции, то IDE запустит копию командного процессора Windows, а в нем наше консольное приложение (рис. 5.8).

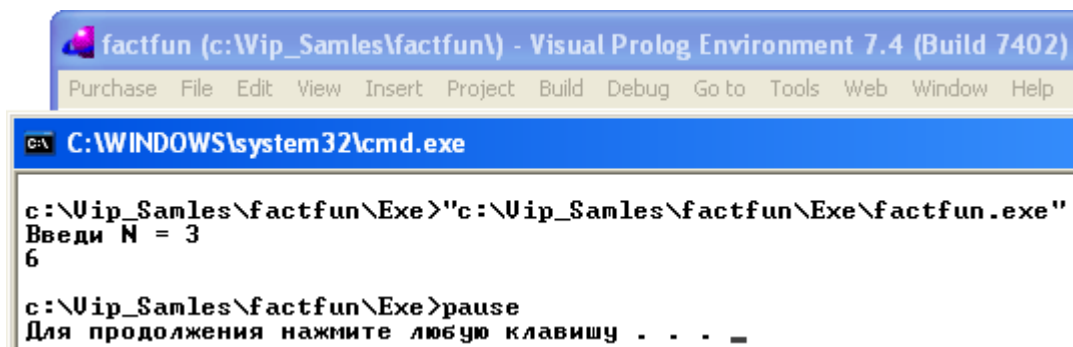


Рис. 5.8. Окно работающего консольного приложения в IDE.

Остается только проверить работу программы, вводя в ответ на запрос программы какое-либо число и получая от нее соответствующий ответ. В этом режиме работы программы последняя строчка с `readChar()` является лишней. Приходится два раза нажимать какую-либо клавишу. Лишнее тут будет необходимо там. А где там?

Если сейчас обратиться в папку, определенную для нашего проекта, то можно обнаружить, что в ней образовалась еще одна папка с именем `Exe`, которая содержит наряду с другими файл `factfun.exe`. Этот файл и является загрузочным модулем нашего проекта, который может выполняться вне среды Visual Prolog'a. Закроем среду и дважды кликнем по пиктограмме `factfun`, приложение запустится в командной строке и будет готово к приему данных с клавиатуры (рис. 5.9). При отсутствии в программе `readChar()` приложение закроется раньше, чем мы увидим ответ.

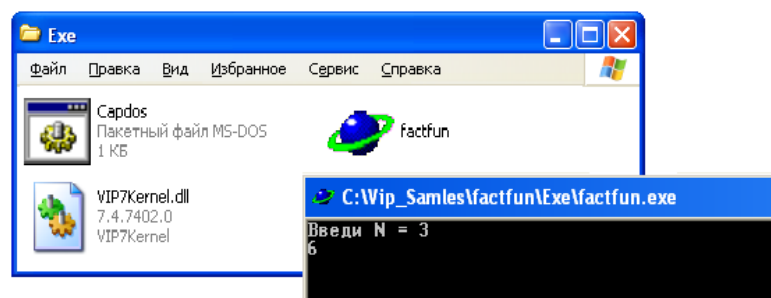


Рис. 5.9. Консольное приложение, запущенное в Windows.

Упражнение 5.2.



- Изменить прототип программного файла до состояния рис. 5.5.
- Разобраться в ее структуре и назначениях стандартных предикатов, используя для этого встроенную справку Visual Prolog.
- Запустить программу на выполнение в IDE и в Windows.
- Реализовать базу знаний likes(<кто>, <что>), выполнить ряд запросов к ней, строя разные варианты текстовых интерфейсов, как в примерах для SWI-Prolog и Turbo-Prolog.

5.3. Разработка GUI проекта на Visual Prolog'е

В этом разделе рассмотрим решение той же задачи, но реализуем ее в виде оконного приложения, которое сможет работать в среде Windows. При реализации этого проекта, как будет видно далее, мало что останется от чистого Prolog'а. В основном это визуальное программирование в среде Visual Prolog'а. При разработке проекта создания полноценного Windows приложения надо выполнить набор следующих действий:

- создать в среде Visual Prolog'а создать новый GUI проект;
- создать для взаимодействия с пользователем некоторую экранную форму, разместить на этой форме элементы управления, определить необходимые при реализации конкретной задачи свойства, как для самой формы, так и для каждого ее элемента управления;
- обеспечить возможность запуска этой формы из основного окна создаваемого приложения;
- создать внутри проекта класс, методы которого будут осуществлять необходимую обработку информации;
- определить события, возникающие в элементах управления формы, которые будут вызывать те или иные методы.

Применительно к задаче вычисления факториала сформированная выше последовательность действий может быть конкретизирована. Пусть требуется разработать Windows приложение, которое представляет собой стандартное окно с панелью управления и главным меню (рис. 5.10).

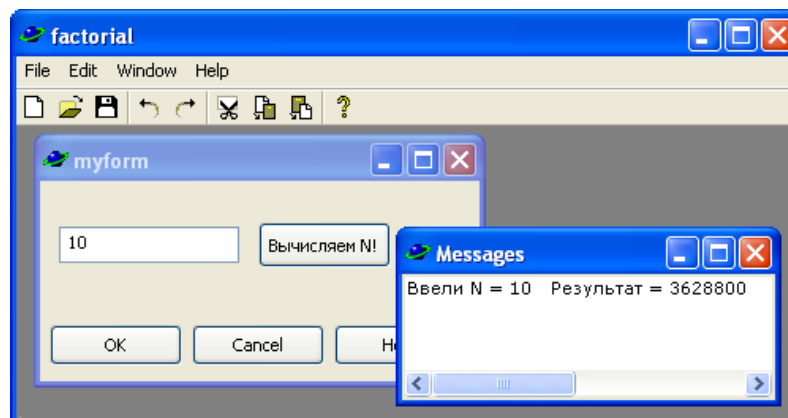


Рис. 5.10. Проект разрабатываемого оконного приложения.

Это приложение должно при выборе опции главного меню File → New открывать экранную форму. На форме должны присутствовать два элемента управления: поле ввода и кнопка. Для кнопки следует определить событие нажатия, по которому запускается процедура вычисления факториала числа, введенного в поле ввода. Нужные для этого процедуры надо организовать в виде класса, включаемого в проект.

Рассмотрим реализацию последовательности всех этих действий, в среде Visual Prolog'a.

Создание нового GUI проекта

Создание проекта выполняется точно так же, как и для консольного приложения. После запуска Visual Prolog'a надо выбрать в его меню опцию Project → New и в диалоговом окне Project Settings указать тип проекта и его имя (рис. 5.11).

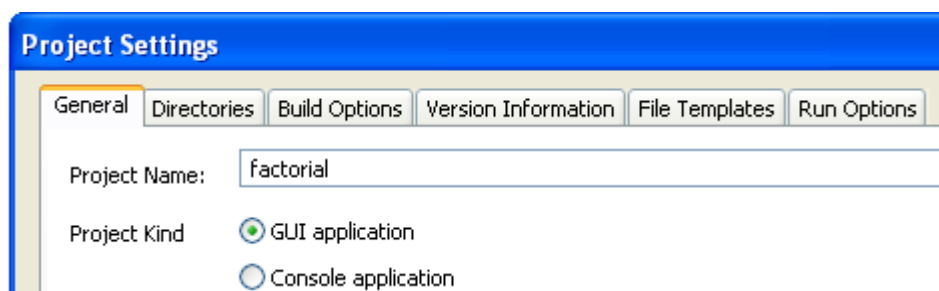


Рис. 5.11. Вид диалогового окна Project Settings.

Система автоматически создает проект, компилирует его, и протокол своих действий выводит в диалоговом окне Messages. В процессе создания сформировался прототип проекта – класс factorial. Его имя совпадает с тем, которое было задано при создании проекта. Обратим внимание, что набор элементов вновь сформированного проекта в дереве проектов во многом совпадает со случаем консольного проекта (рис. 5.12).

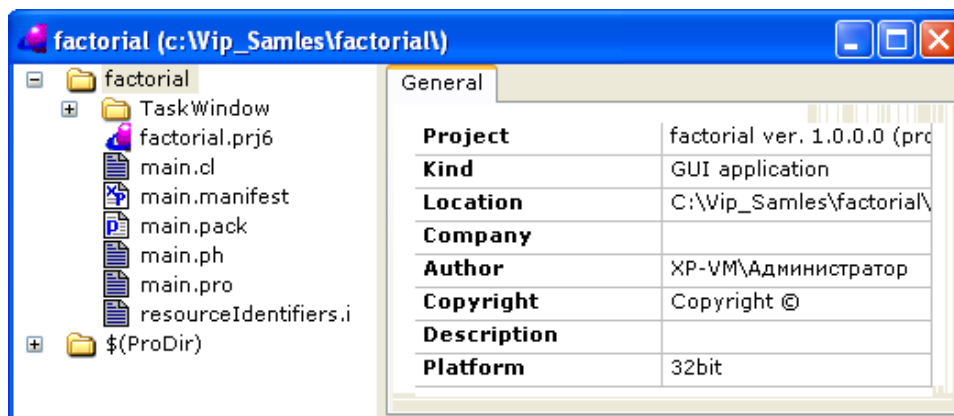


Рис. 5.12. Вид дерева проекта factfun.

Существенным отличием является присутствие в составе GUI проектов узла TaskWindows. Если в дереве проекта раскрыть этот узел (рис. 5.13), то можно увидеть большое количество файлов, которые в Visual Prolog'e служат для поддержки графических приложений.

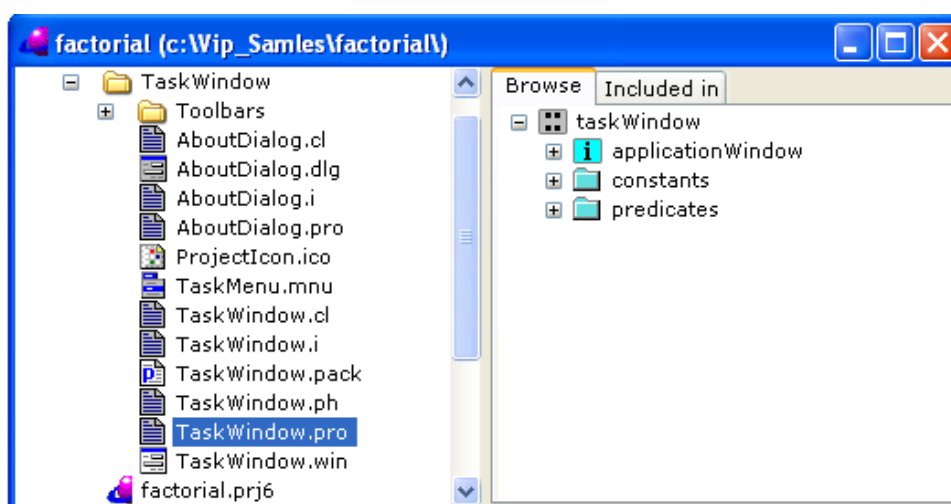


Рис. 5.13. Объекты Visual Prolog'a для поддержки GUI проектов.

Среди этих файлов присутствуют файлы из predeterminedных в Visual Prolog'e категорий:

- .dlg файл – содержит окно диалог (dialog);
- .frm файл – содержит экранную форму (form);
- .win файл – содержит окно, которое может быть окном приложения или обычным окном класса window из PFC GUI (window);
- .mnu файл – содержит настройку и конфигурацию меню (menu);
- .ico файл – содержит иконку приложения(icon)

Можно в проектах встретить также следующие типы файлов:

- .ctl файлы – содержат описание элементов управления (controls);
- .tb файлы – содержат панели инструментов (toolbars);
- .cur файлы – содержат курсоры (cursors);

- .bmp файлы – содержат картинки (bitmaps);
- .lib файлы – это библиотеки (libraries).

Упражнение 5.3.



- Запустить созданное приложение, используя Build → Execute.
- Убедиться в его работоспособности: окно перемещается, меняет свои размеры, активизируются опции главного меню (рис. 5.14).
- Проверь, что все опции меню File недоступны и их нельзя выбрать!!!

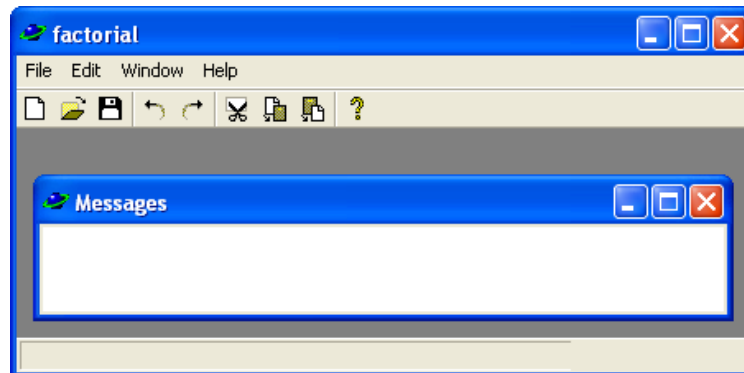


Рис. 5.14. Результат запуска пустого оконного приложения.

Добавление формы в пустой проект

Для того чтобы в проекте создать новую форму необходимо выполнить следующие действия:

- установить в дереве проекта фокус на его вершину – узел factorial,
- выбрать опцию File → New in New Package в главном меню системы и откроется диалоговое окно Create Project Item.

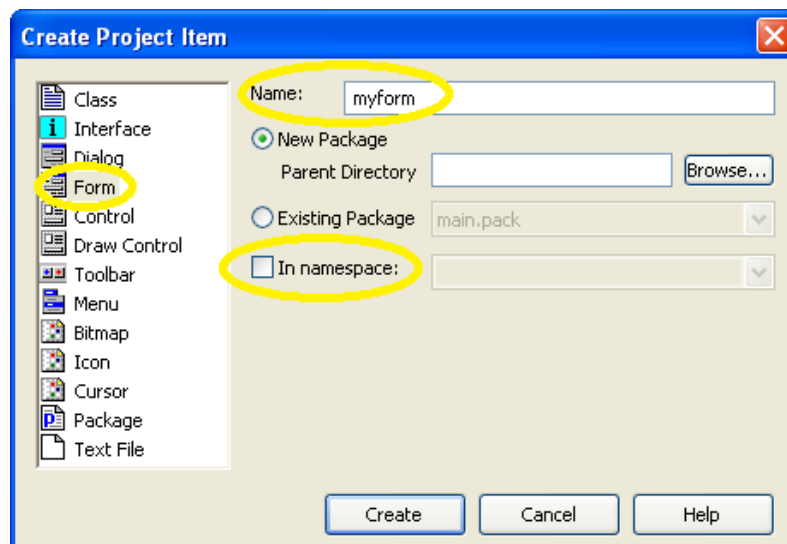


Рис. 5.15. Определение параметров создаваемой формы.

- выбрать в левой панели диалогового окна Create Project Item объект Form, установить параметры, аналогично рис. 5.15.
- После нажатия на кнопку Create диалогового окна Create Project Item, IDE выводит на экран окно прототипа новой формы и еще три окна, предназначенных для создания элементов управления этой формы и настройки их свойств и событий (рис. 5.16).

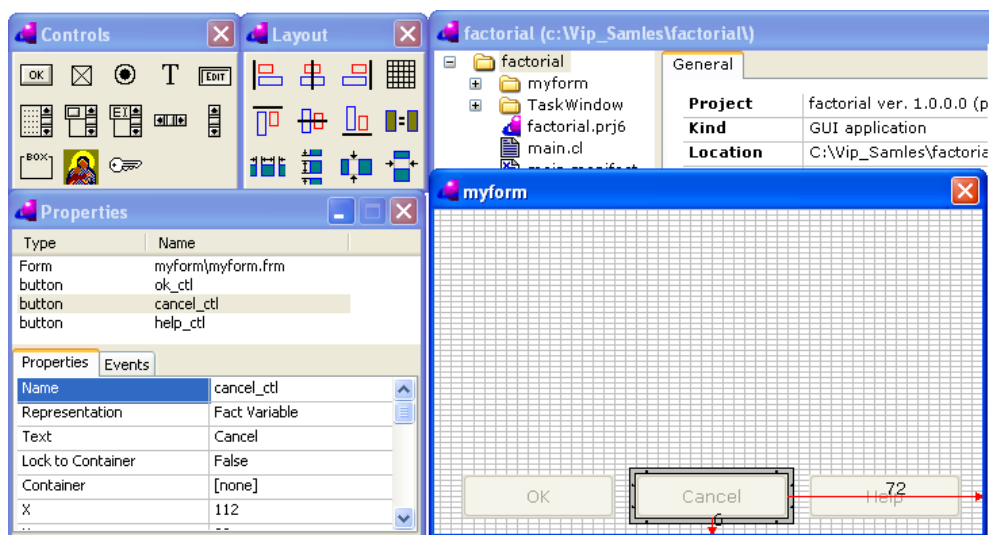


Рис. 5.16. Конструктор экранной формы.

Упражнение 5.4.



- Изменить размеры формы, сделав его немного меньше прототипа.
- Запустить, используя Build → Execute, созданное приложение.
- Убедиться в том, что оно работоспособно, но уже созданную форму открыть невозможно, так как опции меню File все еще недоступны.

Настройка главного меню проекта

Применительно к нашей задаче настройка меню будет состоять всего из двух действий: открытия доступа к опции меню File → New и привязка к этой опции меню события открытия выше созданной формы.

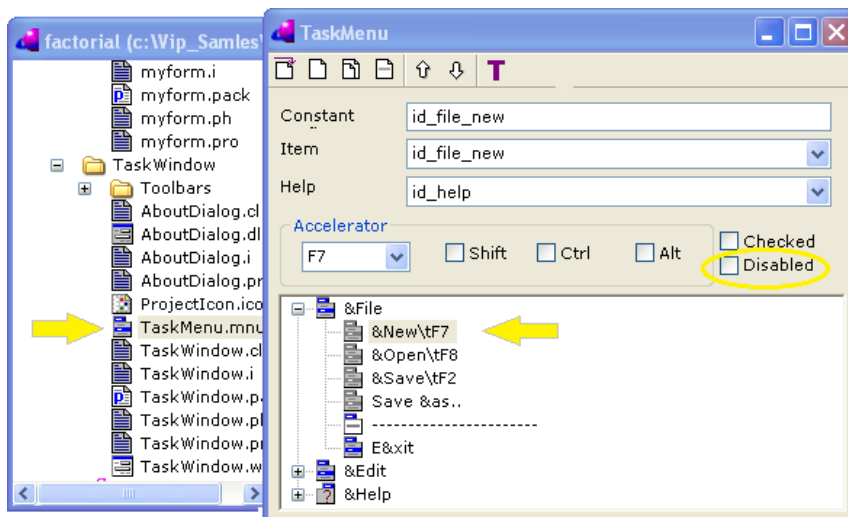


Рис. 5.17. Конструктор настройки меню.

Для выполнения этих операции открытия доступа к опции меню главного меню проекта необходимо:

- Найти в дереве проекта элемент TaskMenu.mnu, двойным кликом на нем открыть диалоговое окно TaskMenu.
- Затем, раскрывая узлы дерева, которое отображается в нижней части диалогового окна TaskMenu, найти пункт меню &New/tF7 и убрать флажок из поля Disabled (рис. 5.17).



Упражнение 5.5.

Выполнить включение опции меню, запустить проект на выполнение и убедиться, что опция доступна, но неактивна (рис. 5.18).

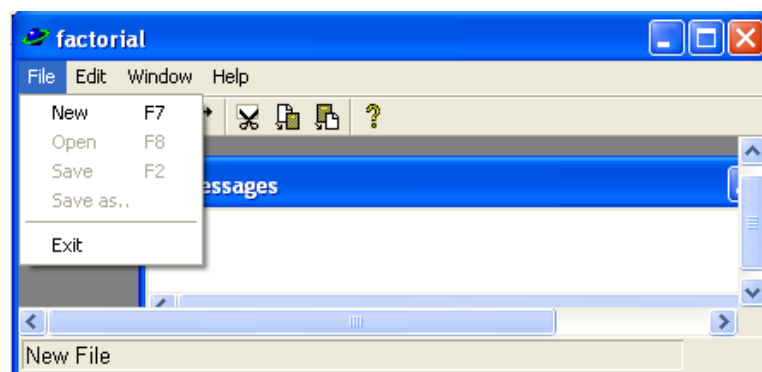


Рис. 5.18. Окно проекта с доступной опцией File → New.

Для привязки к событию выбора пункта меню нужного программного кода необходимо:

- Выделить элемент TaskWindow.win в дереве проекта, двойным кликом открыть диалоговое окно Dialog and Windows Expert.
- Раскрывая узлы дерева Menu, TaskMenu, ..., дойти до узла id_file_new.

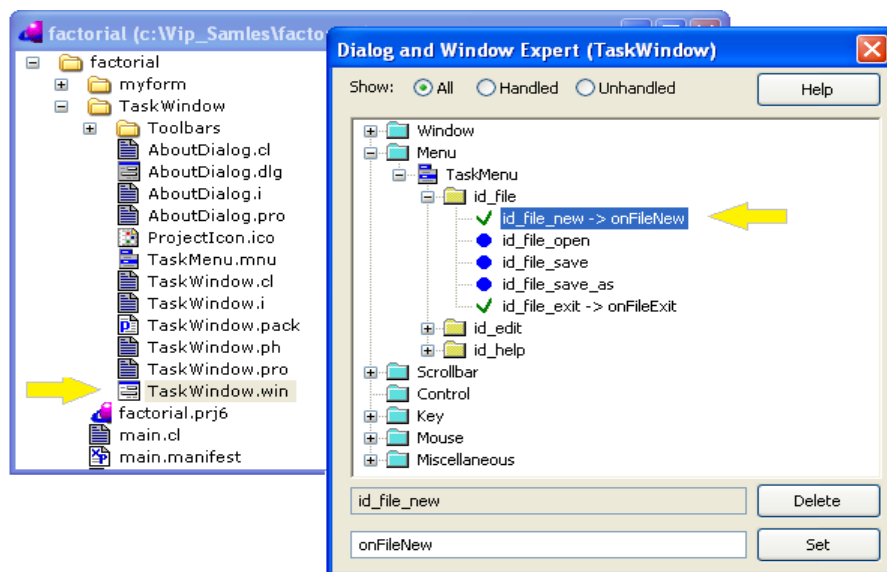


Рис. 5.19. Определение событий, реагирующих на выбор опций меню.

- Нажать кнопку Add. IDE создаст обработчик события onFileNew() и прототип программного кода для обработки этого события.
- Двойной клик в диалоговом окне на строке id_file_new -> onFileNew откроет файл TaskWindow.pro в области определения программного кода обработчика событий onFileNew.
- Заменить прототип кода реальным кодом, который создает экземпляр этой формы и отображает ее на экране (рис. 5.19).

<pre> predicates onFileNew : window::menuItemListener. clauses onFileNew(_Source, _MenuTag). </pre>	<pre> predicates onFileNew : window::menuItemListener. clauses onFileNew(MY_FORM, _MenuTag) :- X = myform:: new(MY_FORM) , X:show(). </pre>
---	---

Рис. 5.19. Код обработчика onFileNew: прототип – слева, реальный – справа.



Упражнение 5.6.

Выполнить описанные действия и убедиться в открытии формы в ответ на выбор File -> New (рис. 5.20).

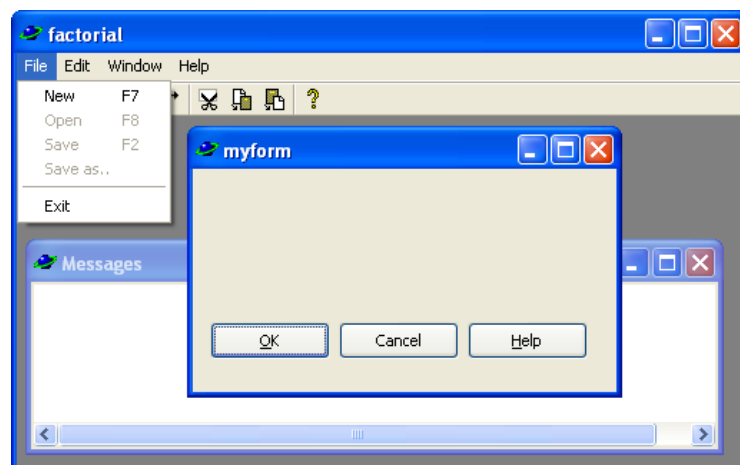


Рис. 5.20. Работающее приложение с прототипом экранной формы.

Разработка экранной формы

Для того чтобы разрабатываемый проект имел практическое значение необходимо наделить экранную форму возможностями по вводу нужных пользователю данных и запуску обработки этих данных. С этой целью надо разместить на форме два элемента управления: поле ввода и кнопку. Чтобы осуществить желаемое, следует:

- в дереве проекта выбрать factorial → myform → myform.frm;
- выполнить двойной клик на myform.frm для того, чтобы открыть конструктор форм (рис. 5.16);
- разместить на форме два элемента управления, изменяя их размеры, местоположение, обратив внимание на возможность использования инструментов выравнивания;
- используя окно Properties, установить желаемый набор свойств для каждого из элементов управления формы, установив: edit_ctl – имя поля ввода, factorial_ctl – имя кнопки и задать ее текст (рис. 5.21).

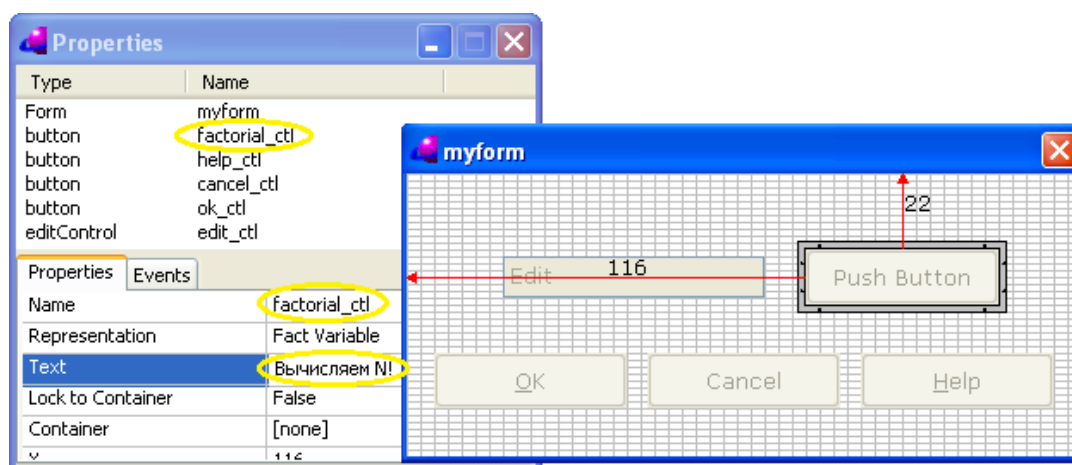


Рис. 5.21. Настройка свойств элементов управления формы.

Для того чтобы размещенная на форме кнопка могла реагировать на свое нажатие, требуется настроить ее обработчика событий. С этой целью следует:

- в окне Properties выбрать factorial_ctl и перейти на вкладку Events;
- в списке событий выбрать ClickResponder, в поле, предназначенном для указания имени процедуры обработки события, щелкнуть мышкой и откроется окно с кодом файла myform.pro (рис. 5.22).

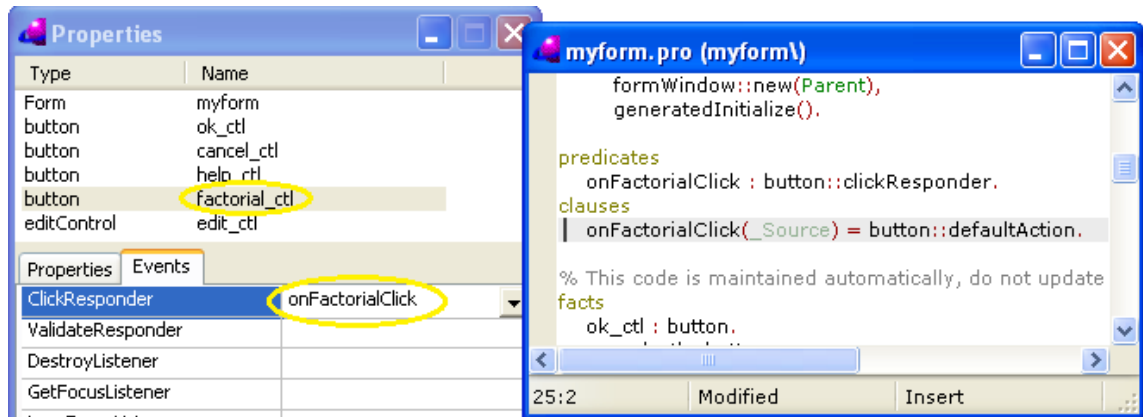


Рис. 5.22. Вкладка Events и прототип кода обработчика событий.

- фокус будет установлен на прототипе процедуры обработки событий onFactorialClick(), которую надо заменить приведенным ниже кодом

```
clauses
  onPushButtonClick(_Source) = button::defaultAction :-
    Vvod = edit_ctl:getText(),
    stdio::write("Ввели N = ", Vvod, "    Результат = ").
%   myclass::calculate(Vvod).
```

Первая строка правой части правила вызывает метод getText() элемента управления edit_ctl для получения содержимого поле ввода, которое унифицируется с переменной Vvod. Вторая строка, используя предикат write класса stdio, обеспечит вывод в стандартный поток. Для простоты мы предполагаем результат отображать не на форме, а выводить в консоль, роль которой будет выполнять стандартное окно системы Messages. Третья строка этого правила пока закомментирована и о ней разговор дальше.

Упражнение 5.7.



Выполнить разработку экранной формы, запустить проект, открыть форму и протестировать ее работу (рис. 5.23).

- Если при запуске проекта появится сообщение типа "Do you want to insert include directive: ...", то нажимайте кнопку Add

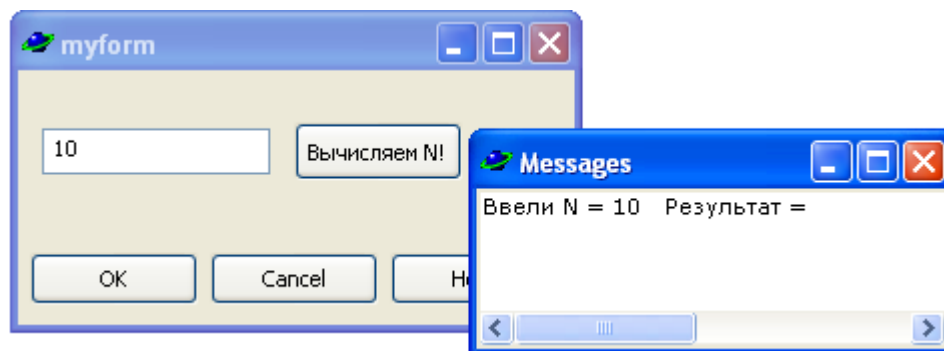


Рис. 5.23. Результат тестирования работы проекта и его экранной формы.

Организация вычислительных процедур проекта

Серьезно говорить о вычислительной, поисковой или интеллектуальной процедуре в данном проекте сложно. Но все-таки у нас есть задача, которая требует вычислить факториал. Именно на ней и будет рассмотрен процесс включения в проект Visual Prolog'a процедур обработки информации.

Все рассматриваемое ранее относится к визуальному проектированию в среде Visual Prolog'a, а собственно мощь декларативных средств Prolog'a по настоящему проявляется только при разработке процедур обработки.

Вернемся к третьей строке кода процедуры обработки нажатия кнопки. Эта строка предполагает, что для вычисления результата вызывается метод `calculate` класса `myclass`, которому передается значение переменной `Vvod`. Однако, ни такого класса, ни такого метода у нас пока не существует. Для создания нового класса надо выполнить действия, которые во многом аналогичны тем, которые выполнялись при создании формы, а именно:

- установить в дереве проекта фокус на его вершину – узел `factorial`,
- выбрать опцию `File -> New in Existing Package` главного меню, откроется диалоговое окно `Create Project Item`.
- в левой панели выбрать объект `Class` и установить параметры, как указано на рис. 5.24;

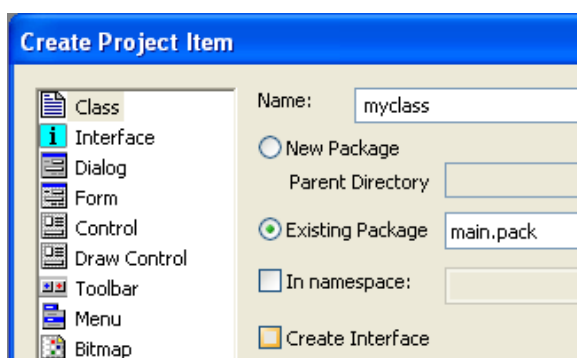


Рис. 5.24. Определение параметров создаваемого класса.

После того, как будет нажата кнопка Create, Visual Prolog создаст ряд прототипов файлов этого класса, которые добавятся в проект и отобразятся в окне дерева проектов. Одновременно с этим откроются два окна с файлами myclass.cl и myclass.pro, которые надо заполнить в соответствии с рис. 5.25.

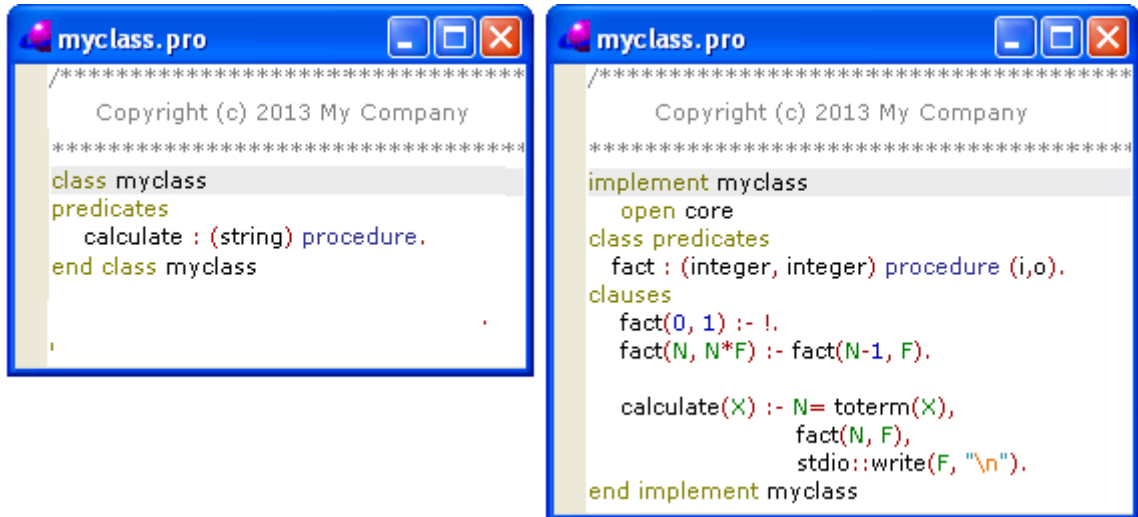


Рис. 5.24. Файлы декларации и описания класса myclass.

Если теперь снять комментарий в процедуре обработки нажатия кнопки и запустить проект на выполнение, то результат будет аналогичен тому, что приведен на рис. 5.10. То есть задача построения GUI приложения решена.

Упражнение 5.8.



- Закончить разработку проекта и запустить его на выполнение, при необходимости провести отладку.
- Изменить некоторые свойства элементов проекта. Например, заголовок формы, начальное значение в поле ввода.
- Доработать проект, чтобы результат выдавался на форме.
- Разобраться с тестовыми программами, входящими в поставку Visual Prolog.

6. PROLOG — ЯЗЫК ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ

6.1. Системы основанные на знаниях

Человек, понимая речь, изображение, образы и иную информацию, для решения возникающих задач использует знания о конкретной предметной области. Для выполнения той же работы компьютером необходимо знания представить в некоторой стандартной форме и составить программу их обработки. При использовании структурных языков программирования необходимые знания помещаются непосредственно в прикладную программу и составляют с ней единое целое.

Однако такой подход затрудняет понимание того, каким образом используются знания, и какую роль они выполняют. Знания, заложенные в программу, и программа их обработки оказываются жестко связанными между собой и представляют возможность получать только те выводы из имеющихся знаний, которые предусмотрены программой их обработки.

В системах, основанных на концепции искусственного интеллекта (ИИ) и инженерии знаний, которые называют системами основанными на знаниях (СОЗ), такая проблема отсутствует. В этих системах функции хранения знаний и функции решения задач разделены подобно базам данных, где системы управления БД обеспечивает автономное хранение данных от программ их обработки (рис. 6.1).

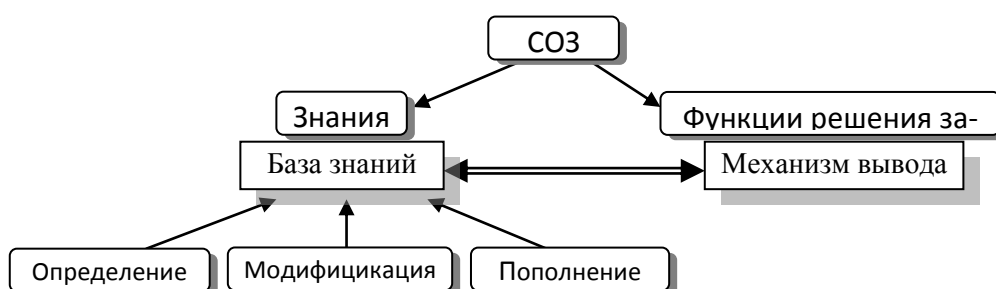


Рис. 6.1. Структура систем, основанных на знаниях.

Принципиальной особенностью систем, основанных на знаниях является тот факт, что в любой из систем этого класса

- знания представляются в конкретной форме базы знаний, которая позволяет знания легко определять, модифицировать и пополнять;

- функции решения задач реализуются автономным механизмом логических выводов, делаемых на знаниях, которые хранятся в базе.

Знания представляют собой совокупность информации и правил вывода (у индивидуума, общества или системы ИИ) о мире, свойствах объектов, закономерностях процессов и явлений, а также правилах использования их для принятия решений. Основное отличие знаний от данных состоит в их структурности и активности. Появление новых фактов или установление новых связей может стать источником изменений в принятии решений.

База знаний (knowledge base) в информатике и исследованиях по ИИ – это особого рода база данных, разработанная для оперирования знаниями (метаданными). База знаний содержит структурированную информацию, покрывающую некоторую область знаний, для использования человеком или кибернетическим устройством с конкретной целью. Современные базы знаний работают совместно с системами поиска информации, имеют классификационную структуру и формат представления знаний.

В современных информационных системах используют различные модели представления знаний – это продукционные, фреймовые или байесовские модели, а также системы на базе семантических сетей, логики предикатов, нечеткой логики и ряд других.

6.2. Логические модели и логическое программирование

Одним из способов представления знаний в информационных системах является язык математической логики, позволяющий формально описывать понятия предметной области и связи между ними.

В отличие от естественного языка, который очень сложен, язык логики предикатов использует только такие конструкции естественного языка, которые легко формализуются. Логика предикатов – это языковая система, которая оперирует с предложениями на естественном языке в пределах синтаксических правил этого языка. Язык логики предикатов использует слова, которые описывают:

- понятия и объекты изучаемой предметной области;
- свойства этих объектов и понятий, а также их поведение и отношения между ними.

В терминах логики предикатов первый тип слов называется термами, а второй тип слов – предикатами. Термы представляют собой средства для обозначения интересующих нас индивидуумов, а предикаты выражают отношения между различными индивидуумами, которые обозначаются с помощью термов.

Логическая модель представляет собой множество предложений, выражающих различные логические свойства именованных отношений.

Логическое программирование – это подход, при котором пользователь описывает предметную область совокупностью предложений в виде логических формул, а кибернетическое устройство, манипулируя этими предложениями, строит необходимый для решения задач вывод.

6.3. Простейшие конструкции языка предикатов

Одной из простейших конструкций логики предикатов является терм. Терм – это знак (символ) или комбинация знаков (символов), являющаяся наименьшим значимым элементом языка. К термам относятся константы, переменные и функции. Константы обозначают конкретные объекты реального мира. Переменные используются для обозначения некоторого из возможных объектов реального мира или их совокупности. Обычно их обозначения начинаются с заглавной буквы.

Структуры – представляют собой последовательность разделенных запятыми констант или переменных, которые заключены в круглые скобки и следуют за функциональным символом (функтором). Функторы обозначают операторы, которые после воздействия на объект возвращают некоторое значение. Пример записи структур: сумма (1,2); +(1,2); удвоить (X), а примерами работы с ними и их унификации могут быть:

```
?- +(2, 3) = 2 + 3.  
true.  
  
?- +(2, *(5,6)) = +(X, *(5, Y)).  
X = 2,  
Y = 6.  
  
?- Result is +(2, *(5,6)).  
Result = 32 ?
```

Предикат – это логическая функция, которая выражает отношение между своими аргументами и принимает значение «истина», если это отношение имеется, или «ложь», если оно отсутствует. Заключенная в скобки последовательность из n термов, перед которой стоит предикатный символ, называется n-местным (или n-арным) предикатом, который принимает значения «истина» или «ложь» в соответствии со значением термов, являющимися его аргументами. Примеры записи предикатов:

```
является ( ласточка, птица ).  
отец (X, "Петр").
```

Такого типа предикаты получили название атомарных предикатов и соответствуют наиболее простым предложениям разговорного языка – не-

распространенным предложениям. В обычном человеческом языке из нераспространенных предложений с помощью соединительных союзов, местоимений, и других частей речи строят более сложные конструкции – сложные предложения.

6.4. Предикатные формулы

В логике предикатов сложным предложениям естественного языка соответствуют предикатные формулы. Предикатные формулы образуются из атомарных предикатов с использованием логических связок. Эти связки еще называют пропозициональными и выделяют пять типов таких связок, которые приведены в табл. 6.1.

Таблица 6.1

Обозначение и назначение логических связок

Обозначение	\neg	, или \wedge	; или \vee	\rightarrow	\leftrightarrow или \sim
Читается, как ...	НЕ	И	ИЛИ	Если ..., То , Из ... следует	Тогда и только ...
Операция	отрицание	конъюнкция	дизъюнкция	импликация	эквиваленция

В логических выражениях порядок выполнения логических операций задается круглыми скобками. При их отсутствии логические связки имеют следующий приоритет использования: сначала выполняется отрицание ("не"), затем конъюнкция ("и"), после конъюнкции — дизъюнкция ("или") и в последнюю очередь — импликация.

В логике высказываний показано, что операцию импликации можно выразить через дизъюнкцию и отрицание: $A \rightarrow B = \neg A \vee B$, а операцию эквивалентности можно представить через операции отрицания, дизъюнкции и конъюнкции: $A \leftrightarrow B = (\neg A \vee B) \wedge (\neg B \vee A)$.

Таким образом, операций отрицания, дизъюнкции и конъюнкции вполне достаточно, чтобы описывать и обрабатывать любые логические высказывания. Что касается логического программирования, то в нем наиболее часто используют связки «И», «НЕ» «ЕСЛИ». Тогда предикатная формула, соответствующая сложному предложению, может иметь вид, например,

является(ласточка, птица) \leftarrow имеет(ласточка, крылья),
владеет(ласточка, гнездо).

где является/2, имеет/2, владеет/2 – это атомарные предикаты, а «, \rightarrow » и « \leftarrow » представляют собой логические связки.

Однако приведенная конструкция предикатной формулы позволяет делать утверждение не только о конкретном индивидууме, которым является

ласточка, но и обо всех индивидуумах из класса птиц, используя для этого вместо констант переменные:

|| является (X, птица) \leftarrow имеет (X, крылья) , владеет (X, гнездо)

Используя переменные вместо конкретных имен, мы приходим к более общим понятиям кортежа длины n, предиката и логической формулы. Однако предикат, который содержит переменные, например,

|| имеет (X, крылья)

не может быть однозначно оценен, так как невозможно определить ложь он или истина, Его значение определяется после подстановки в переменную некоторой константы.

Однако, иногда можно определить значения предиката, не делая подстановок, используя кванторы общности (\forall) и существования (\exists), которые обозначают «для всех» и «существует, по крайней мере, одно». Тогда приведенная выше логическая формула будет записана в виде:

($\forall X$) [является (X, птица) \leftarrow имеет (X, крылья) ,
владеет (X, гнездо)]

и соответствуют предложению, которое может читаться как: «любой X является птицей, если этот X имеет крылья и владеет гнездом».

Кванторное логическое высказывание с квантором всеобщности вида $\forall x A(x)$ – истинно только тогда, когда для каждого объекта x из заданной совокупности высказывание $A(x)$ истинно. Логическое высказывание с квантором существования $\exists x A(x)$ – истинно только тогда, когда в заданной совокупности существует объект x, такой, что высказывание $A(x)$ истинно.

Кванторы \forall и \exists могут использоваться и для любого числа переменных, а при их комбинации очень важен порядок их использования. Рассмотрим их различное использование, и соответствующее семантическое толкование высказываний на примере использования двуместного предиката любит/2, который описывает отношение «X любит Y»:

$\forall X \forall Y$ любит (X, Y) – все любят всех;

$\exists X \forall Y$ любит (X, Y) – существует такой X, который любит всех;

$\forall X \exists Y$ любит (X, Y) – любой X любит хотя бы одного Y;

$\exists X \exists Y$ любит (X, Y) – существует такой X, который кого-либо любит;

$\exists Y \exists X$ любит (X, Y) – существует такой Y, которого кто-то любит;

$\forall Y \exists X$ любит (X, Y) – каждого Y любит хотя бы один X.

6.5. Определение правильно построенной формулы

Комбинируя логические связки и кванторы можно рекурсивно определить составную формулу логики предикатов, называемую правильно построенной формулой (далее просто ППФ или логическая формула). Если говорить применительно к естественному языку, то ППФ описывает обычное предложение общего вида:

1. Термом является либо константа, либо переменная, либо кортеж из n термов, перед которым стоит функтор.

2. Предикат – кортеж из n термов, перед которым стоит предикатный символ.

3. Атомарный предикат является логической формулой.

4. Если F и G - логические формулы, то (F) , (F,G) , $(F \vee G)$, $(\neg F)$, $(F \rightarrow G)$, $(F \leftrightarrow G)$ – также являются логическими формулами.

5. Если $F(X)$ - логическая формула, то выражения $\forall X F(X)$ и $\exists X F(X)$ являются логическими формулами.

6. Все результаты, получаемые повторением конечного числа раз пунктов с 1 по 6, являются правильными логическими формулами.

Множество предложений, которые построены согласно этим правилам, образуют язык логики предикатов первого порядка, который, являясь формализованным аналогом обычной логики, дает возможность строго рассуждать об истинности и ложности утверждений и об их взаимосвязи. В частности, о логическом следовании одного утверждения из другого, или, например, об их эквивалентности.

Рассмотрим классический пример формализации утверждений естественного языка в логике первого порядка. “Все люди смертны” — этими словами начинается самый знаменитый из силлогизмов, гласящий далее: “Сократ — человек” и “следовательно, Сократ смертен”. Используя определения правильно построенной формулы, первое предложение можно записать в виде $\forall X(\text{человек}(X) \leftarrow \text{смертен})$, а второе предложение в виде факта человек(“Сократ”).

6.6. Логический вывод

Логический вывод – это процесс получения из множества правильно построенных логических формул $\{S_i\}$ некоторой новой ППФ (s) путем применения одного или нескольких правил вывода.

Одним из таких правил является правило *modus ponens*, которое используется в исчислении высказываний. Его еще называют правилом отделения или гипотетическим силлогизмом. Простой силлогизм — это рассуждение мысли, которое состоит из трёх простых атрибутивных высказы-

ваний: двух посылок и одного заключения. Примером силлогизма может быть только что рассмотренный пример:

Всякий человек смертен	(бóльшая посылка)
Сократ — человек	(меньшая посылка)
<hr/>	
Сократ смертен	(заключение)

Правило вывода *modus ponens* позволяет от утверждения условного высказывания $A \rightarrow B$ и утверждения его основания A (антецедента) перейти к утверждению следствия B (консеквенту). Другими словами, если A и $A \rightarrow B$ — выводимые формулы, то B также выводима. Формат записи этого правила вывода будет иметь вид:

$$\frac{A, A \rightarrow B}{B}$$

Следует отметить, что рассмотренное правило является частным случаем правила резолюций, которое в свою очередь относится к методу доказательства теорем через поиск противоречий. Правило резолюций, последовательно применяемое для списка резольвент, позволяет ответить на вопрос, существует ли в исходном множестве логических выражений противоречие.

6.6.1. Правило резолюции для простых предложений

Наиболее простой метод логического вывода использует только одно правило вывода, называемое *резолюцией*, которое применяют к логическим формулам вида:

факт: A
отрицание: $\neg (A_1, \dots, A_n)$
импликация: $A \leftarrow B_1, \dots, B_m$,

где A_i ($i = 1, n$) и B_j ($j = 1, m$) - произвольные предикаты. Рассмотрим наиболее простую из форм резолюции для случая всего лишь двух правильно построенных формул $S = \{ S1, S2 \}$ вида:

$S1$ (отрицание): $\neg A$
 $S2$ (импликация): $A \leftarrow B$,

в которых предикат A из $S1$ совпадает с предикатом A левой части $S2$. В результате одного шага вывода из $S1$ и $S2$ будет получена новая ППФ вида:

s (резольвента): $\neg B$

На этом шаге вывода правильно построенные формулы S1 и S2 называются *родительскими предложениями*, а S – *резольвентой*, которая получается в результате применения резолюции к S1 и S2. Резолюция в этом простейшем случае соответствует правилу вывода modus tollens, которое записывается в виде:

$$(*) \quad \frac{\neg A, A \leftarrow B}{\neg B}$$

и соответствует следующему умозаключению:

$$\begin{array}{l} \text{Допуская, что: не } A \\ \text{и} \quad \underline{A \text{ если } B} \\ \text{Выводим:} \quad \text{не } B \end{array}$$

В еще более простом случае, когда логическая формула S1 представляет собой отрицание, а логическая формула S2 – факт:

$$\begin{array}{l} S1 \text{ (отрицание): } \neg A \\ S2 \text{ (факт): } A \end{array}$$

применение правила резолюции даст резольвенту в виде пустого отрицания

$$S : \square,$$

которое означает противоречие. Этот шаг вывода может быть записан в следующем виде:

$$\frac{\neg A, A}{\square} \quad (**)$$

и соответствует следующему умозаключению:

$$\begin{array}{l} \text{Допуская, что: не } A \\ \text{и} \quad \underline{A} \\ \text{Выводим} \quad \text{противоречие.} \end{array}$$

6.6.2. Правило резолюции для сложных предложений

Реальные логические модели содержат значительно более сложные предложения. Так отрицание могут содержать несколько предикатов, так же как и правые части импликаций. Поэтому более общим является случай, когда родительские предложения имеют вид:

$$\begin{array}{l} S1: \neg (A_1, \dots, A_k, \dots, A_n) \\ S2: A_k \leftarrow (B_1, \dots, B_m) \end{array} \quad (\text{где } 1 < k < n)$$

Здесь некоторый предикат A_k из отрицания S_1 совпадает с предикатом левой части S_2 . В этом случае один шаг вывода заменяет предикат A_k в логической формуле S_1 на правую часть логической формулы S_2 и в качестве резольвенты получают отрицание вида:

$$\left\| \begin{array}{l} S: \quad \neg (A_1, \dots, A_{k-1}, B_1, \dots, B_m, A_{k+1}, \dots, A_n) \end{array} \right.$$

Данное правило вывода для сложных предложений проиллюстрируем содержательным примером. Например,

$$\begin{array}{ll} \text{допуская, что} & \text{Не (темно и зима и холодно)} \\ \text{и что} & \text{Зима если Январь} \\ \hline \text{выводим, что} & \text{НЕ (темно и январь и холодно)} \end{array}.$$

Рассмотрим случай, когда существует две логические формулы, в которых S_1 имеет тот же вид, что и ранее, а логическая формула S_2 представляет собой факт:

$$\left\| \begin{array}{l} S_1: \quad \neg (A_1, \dots, A_k, \dots, A_n) \\ S_2: \quad A_k \end{array} \right.$$

Причем факт A_k аналогичен одному из предикатов, входящих в S_1 . Тогда один шаг вывода заключается в том, что он только вычеркивает предикат A_k из S_1 и получает резольвенту вида

$$\left\| \begin{array}{l} S: \neg (A_1, \dots, A_{k-1}, A_{k+1}, \dots, A_n) \end{array} \right.$$

и данный шаг вывода можно иллюстрировать следующим примером

$$\begin{array}{ll} \text{допуская, что} & \text{Не (темно и зима и холодно)} \\ \text{и что} & \text{Зима} \\ \hline \text{выводим, что} & \text{НЕ (темно и холодно)} \end{array}.$$

6.6.3. Простая резолюция сверху вниз

Рассмотренные выше правила применяются на каждом шаге вывода только к двум родительским предложениям. Вместе с тем описание любой области знания содержит множество ППФ. Рассмотрим процесс логического вывода для примера, когда знания выражаются двумя предложениями.

$$\left\| \begin{array}{l} S_2: \text{получает (студент, стипендию)} \leftarrow \\ \quad \text{сдаёт (успешно, сессию, студент)} \\ S_3: \text{сдаёт (успешно, сессию, студент)} \end{array} \right.$$

Задача, которую надо решить на этом множестве ППФ состоит в том, чтобы ответить на вопрос: "Получает ли студент стипендию?". Когда будет использоваться обычная система логического вывода, то такой вопрос представляется запросом в виде отрицания

|| $S_1: \neg$ получает (студент, стипендию)

и задача системы состоит в том, что она должна отвергнуть это отрицание при помощи других предложений, демонстрируя, что данное допущение ведет к противоречию. Этот подход часто применяется в математике и называется *доказательством от противного*.

Теперь представим себе, что исходная логическая модель, составленная из трех предложений S_1, S_2, S_3 поступает на вход системы логического вывода компьютерной системы.

Шаг 1. Система на первом шаге применит правило вывода (*) к родительским предложениям S_1 и S_2 и получит резольвенту:

|| $s: \neg$ сдает (успешно, сессию, студент)

Шаг 2. Используя правило (**) к ППФ s и S_3 система выводит противоречие, а именно $s': \square$

Таким образом, для доказательства противоречивости S_1, S_2 и S_3 оказалось достаточно двух шагов вывода. Если считать, что S_2 и S_3 не противоречат друг другу, то они совместно противоречат S_1 или другими словами подтверждают предложение: получает (студент, стипендию). И в этом случае ответом на исходную задачу является «Истина (true/ДА)».

Логический вывод, который порождает последовательность отрицаний, такую как s_1, s, s' в данном примере, называется *резолюцией сверху вниз*.

6.6.4. Общая резолюция сверху вниз

Однако в большинстве случаев структура предложений имеет более сложный вид. В частности, предикаты и логические формулы в качестве термов могут содержать не только константы, но и переменные, и функции. В этих условиях несколько модифицируется и сама процедура логического вывода. Чтобы получить самые общие представления об общей резолюции сверху вниз рассмотрим для примера два родительских предложения вида:

|| $S_1: \neg$ получает (студент, Y)
|| $S_2: \text{получает}(X, \text{стипендию}) \leftarrow \text{сдает}(\text{успешно}, Z, X)$

К ним непосредственно уже нельзя применить правило резолюции, так как они не содержат одинаковых предикатов в левой части импликации и в отрицании. Данные предложения содержат три переменных X, Y, Z , которые неявно универсально квантифицированы. Рассмотрим первое предложение S_1 , которое утверждает, что:

для всех Y (студент не получает Y)

Причем выражение «для всех» понимается как «для всех индивидуумов, из какой либо области, выбранной для интерпретации предложений». При интерпретации S_1 и S_2 , по крайней мере, один индивидуум Y будет связан с именем «стипендия» и поэтому непосредственным следствием S_1 является более конкретное предложение:

$S_1^1: \exists \text{ получает}(\text{студент}, \text{ стипендию})$

Аналогично рассматривается S_2 на области интерпретации S_1 и S_2 и, выбирая для X , индивидуум с именем «студент», получаем более конкретное предложение:

$S_2^1: \text{получает}(\text{студент}, \text{ стипендию}) \leftarrow \text{сдает}(\text{успешно}, Z, \text{ студент})$

Теперь имеем два предложения S_1^1 и S_2^1 , которые удовлетворяют условию применимости правила резолюции. Это условие предполагает наличие одинаковых предикатов в левой части импликации и в отрицании. Поэтому резольвентой логических формул S_1^1 и S_2^1 , после применения к ним правила (*) будет:

$s: \exists \text{ сдает}(\text{успешно}, Z, \text{ студент})$

Предикат получает (студент, стипендию) называется *общим примером* родительских предикатов

$\text{получает}(X, \text{ стипендию})$
 $\text{получает}(\text{студент}, Y)$

и получен с помощью унификатора вида $\Theta = \{X := \text{студент}, Y := \text{стипендия}\}$.

6.6.5. Унификаторы и примеры унификации

Унификатором называется множество присваиваний вида

$$\Theta = \{X_1 := t_1, \dots, X_n := t_n\}$$

где X_i - переменная, а t_i – терм, применение которых к двум выражениям дает одинаково общие примеры.

На практике унификаторы определяют, сравнивая по очереди соответствующие аргументы предикатов и выписывая те присваивания термов переменным, которые сделали бы эти аргументы одинаковыми. Для пояснения этого процесса рассмотрим ряд примеров унификации, которые приведены в табл. 6.2.

Таблица 6.2

Примеры унификации

Родительские предложения	Унификатор	Общий пример
$p(5), p(5)$	Θ - пустое множество (не заменяется ни одна переменная)	$p(5)$
$p(x), p(5)$	$\Theta = \{x:=5\}$	$p(x)\Theta = p(5)\Theta = p(5)$
$p(x), p(y)$	$\Theta = \{x:=y\}$	$p(y)$
$p(x, y), p(5, x)$	$\Theta = \{x:=5, y=x\} =$ $= \{x:=5, y:=5\}$	$p(x,y)\Theta = p(5,x)\Theta =$ $p(5,5)$
$\neg p(5, x)$ $p(x, y) \leftarrow q(x)$	$\Theta = \{x:=5, y:=5\}$	$p(5,5)$ резольвента: $s: \neg q(x)\Theta = \neg q(5)$

6.7. Решение задач и извлечение ответа

Решение задачи с использованием логического программирования разбивается на 3 этапа:

На первом этапе необходимо сформулировать наши знания и допущения о предметной области в виде множества ППФ.

На втором этапе нужно выразить конкретную задачу, поставленную на предметной области, как запрос об одном или нескольких отношениях. Обычно запрос ставится как исходное отрицание.

Составлением допущений и исходного запроса завершается работа программиста, цель которой – сформулировать задачу.

Третий шаг выполняется компьютером, который пытается решить задачу, строя доказательство от противного. Он строит вывод сверху вниз, начиная с исходного отрицания, и порождает последовательность отрицаний D_1, D_2, \dots, D_n . Если может быть построен вывод, который заканчивается отрицанием, то есть $D_n = \square$ (противоречие), то этот вывод, называемый успешным и сразу дает решение поставленной задачи.

В качестве примера рассмотрим последовательность выполнения всех этих этапов на примере вычислительной задачи нахождения значения факториала некоторого числа.

На первом этапе необходимо в виде ППФ сформулировать наши знания о вычислении факториала, которые можно представить двумя предложениями:

$S_1 : \text{факториал}(0, 1)$

$S_2 : \text{факториал}(X, F) \leftarrow X > 0, N = X - 1, \text{факториал}(N, Y), F = Y * X$

На втором этапе конкретную задачу, например, вычисление значения факториала числа 3 ($3! = 1 * 2 * 3$), формулируем в виде запроса как исходного отрицания:

$D_1 : \neg \text{факториал}(3, Z)$

На третьем этапе система логического вывода выполнит приведенную в табл. 6.3 последовательность действий, применяя на каждом из шагов соответствующие правила резолюции.

Таблица 6.3

Вывод на основе резолюции

Шаг	Родительские предложения	Унификатор	Отрицание (резольвента)
1	D_1, S_2	$\Theta = \{X:=3, F:=Z, Y:=F/X\} = \{N:=2, Y:=z/3\}$	$D_2 : \neg \text{факториал}(2, Z/3)$
2	D_2, S_2	$\Theta = \{X:=2, F:=Z/3, Y:=F/X\} = \{N:=1, Y:=Z/6\}$	$D_3 : \neg \text{факториал}(1, Z/6)$
3	D_3, S_2	$\Theta = \{X:=1, F:=Z/6, Y:=F/X\} = \{N:=0, Y:=Z/6\}$	$D_4 : \neg \text{факториал}(0, Z/6)$
4	D_4, S_1	$\Theta = \{Z/6:=1\} = \{Z:=6\}$	\square

Полученное на шаге 4 противоречие подтверждает отрицание D_1 , а стало быть, вывод является успешным и дает решение: факториал(3,z), с унификатором $\Theta = \{Z:=6\}$. Ответ, который выдаст система логического вывода, будет: $Z:=6$

ПРИЛОЖЕНИЕ

Приложение 1. Запуск Turbo-Prolog в ОС Windows 7

Версии Windows стали настолько продвинутыми, что уже десяток лет не поддерживают DOS-приложения. Все версии после 9x (93,95,98 и т.д.) работают на совершенно другой технологии под названием NT (New Technology). Поэтому для обеспечения совместимости с DOS программами Microsoft некоторое время поддерживала систему под названием NTVDM (NT Virtual DOS Machine), которая эмулирует DOS. Однако, начиная с Windows Vista и 7, эту поддержку Microsoft удалила. Поэтому для запуска Turbo Prolog на любой версии Windows Vista или 7 нужны специальные средства.

Microsoft предлагает запускать DOS-программы в виртуальной среде Windows XP mode. Этот подход работает, но данная технология слишком громоздка, запуск и завершение программ занимает много времени. Однако, есть более удобное решение сторонних разработчиков — DOSBox.

Эксперименты показали, что в DOSBox та же программа работает быстрее на четверть. Установка и настройка занимает значительно меньше времени. Загрузить сам DOSBox и его русскую локализацию под Windows можно с официального сайта разработчика (рис. п1.1)

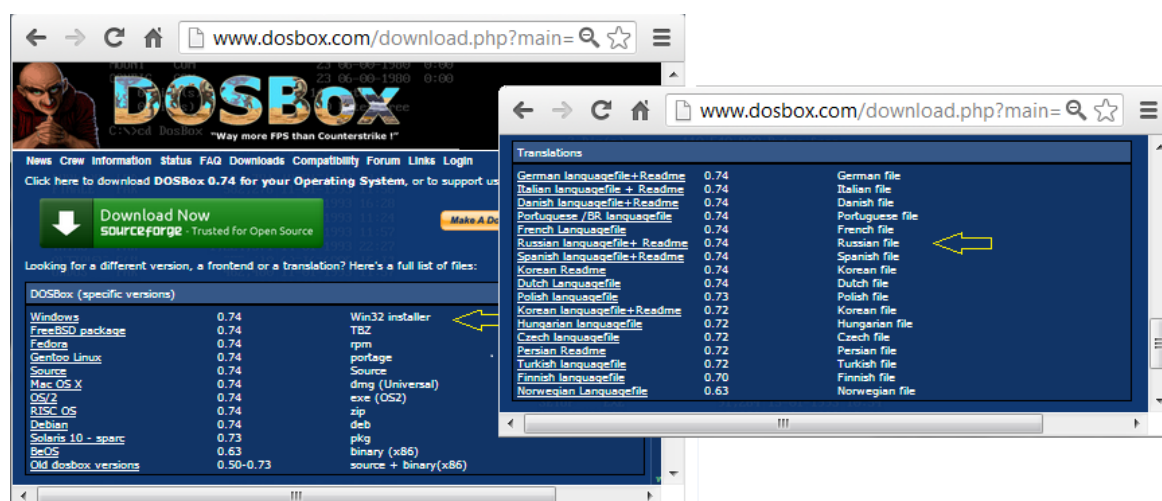


Рис. п1.1. Страницы загрузок сайта www.dosbox.com.

Установка DOSBox выполняется стандартным для Windows способом, после окончания которой, экран системы и ее главное меню будет иметь вид, аналогичный представленному на рис. п1.2. Обратите внимание на опцию «DOSBox 0.74 Options». Она служит для изменения конфигурации DOSBox. Выбор этой опции вызывает на редактирование файл dosbox-0.74.conf, о котором речь пойдет немного позднее.

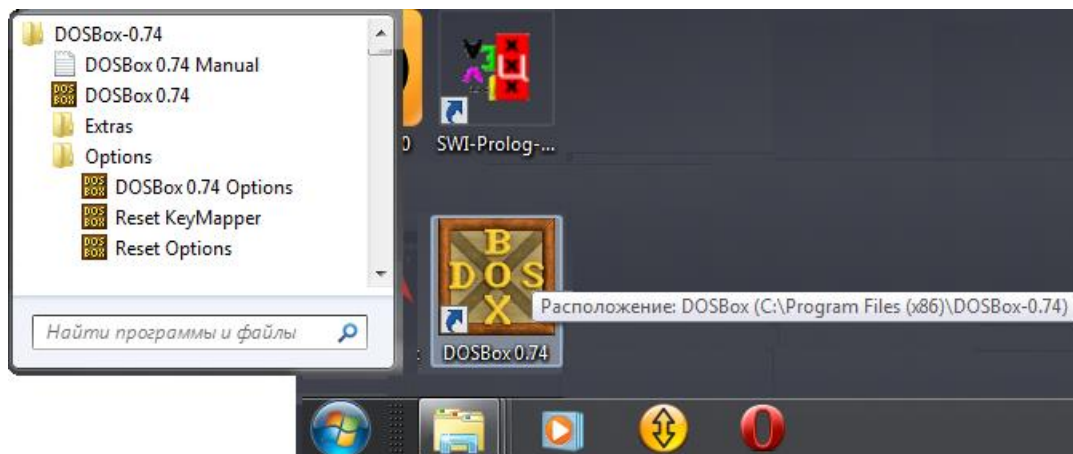


Рис. п1.2. Главное меню и ярлык DOSBox в Windows 7.

Запуск DOSBox

После щелчка по ярлыку DOSBox откроются два окна программы. В первом выводятся служебные сообщения, а во втором – командная строка DOS, с которой мы пока и будем работать. Сейчас DOSBox поддерживает многие стандартные команды DOS, так что люди знакомые с DOS могут чувствовать себя как дома. Обратите внимание на подсказку командной строки: при старте в качестве текущего диска указан диск Z.

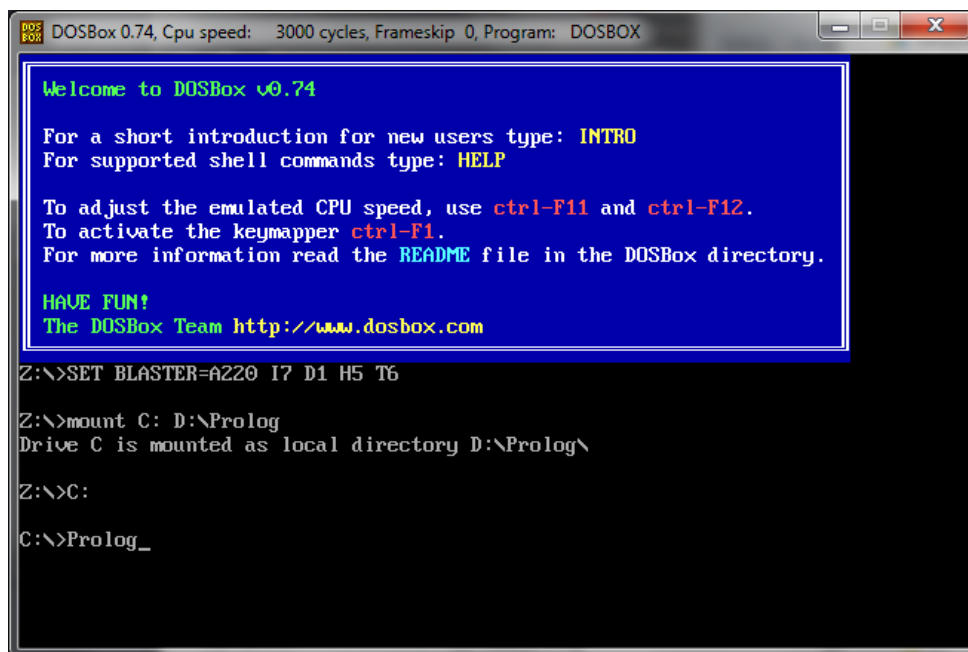


Рис. п1.3. Окно с командной строкой DOSBox.

Он представляет собой виртуальный диск в памяти ПК, на котором находятся загрузочные и служебные файлы эмулятора. Никакого диска C:\ по умолчанию в программе нет. Это сделано для того, чтобы не причинить вреда данным пользователя. DOSBox использует подключение дисков, причём, вместо диска можно задать название каталога. Если файлы Turbo Prolog'a на основном компьютере расположены в папке D:\Prolog, то диск C:\ в среде DOSBox может означать и эту папку. При этом программа, запущенная из папки D:\Prolog будет думать, что она работает с корневым каталогом диска C:\, и не сможет испортить остальные данные в случае критической ситуации. Для монтирования диска используют команду.

```
mount [диск_DOSBox] [полный_путь_к_папке_TurboProlog]
```

Тогда для запуска Turbo Prolog'a в среде DOSBox достаточно будет, например, следующей последовательности команд:

```
Z:\> mount c: d:\Prolog
Z:\> c:
C:\> prolog.exe
```

Их выполнение в среде DOSBox приведет к тому, что на экране дисплея компьютера с Windows Vista или Windows 7 появится окно с полноценно работающей DOS-программой (рис. п1.4).

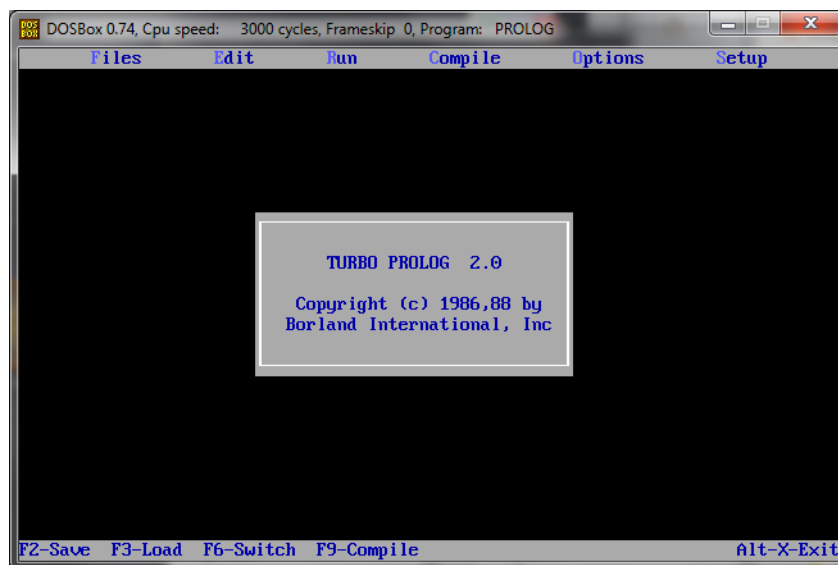


Рис. п1.4. Окно Turbo Prolog'a, запущенного в среде DOSBox.



Замечание.

Не забудьте, что имена файлов и папок в DOS имеют восемь символов на название и три на расширение. Например, папка "Program Files" выглядит как "Progra~1". Поэтому для подключаемых папок и файлов надо иметь соответствующие имена.

Диск, который был примонтирован к DOSBox, также легко может быть и размонтирован (удален) из DOSBox. Для этого используют команду:

```
mount -u [имя_диска_в_DOSBox]
```

Если для запуска Turbo Prolog'a после старта DOSBox вам надоест каждый раз выполнять монтирование диска и запуск программы, то есть возможность автоматизировать это действие. Достаточно воспользоваться опцией «DOSBox 0.74 Options» главного меню Windows. При ее выборе в текстовом редакторе откроется конфигурационный файл DOSBox, в котором надо найти секцию [autoexec] и добавить в нее несколько строк (рис. п1.5).

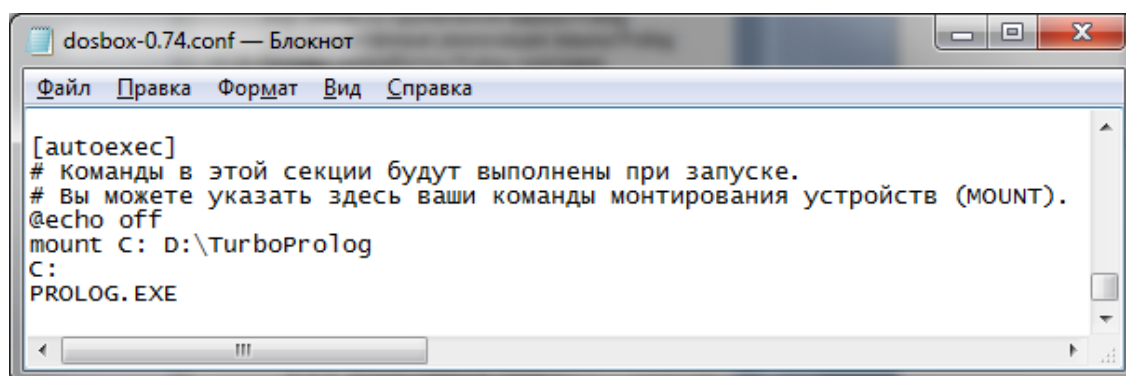


Рис. п1.5. Изменение секции [autoexec] конфигурационного файла DOSBox.

Подключение русского языка

Если вы планируете запускать программы, использующие русский язык, то будут большие проблемы. Поможет в их решение скачанный с сайта разработчика архив локализации DOSBox-russian-lang-074.zip. С его помощью надо изменить конфигурацию DOSBox.

Внимание!!!



- Файлы конфигурации в зависимости от того, как был установлен DOSBox могут находиться либо в папке с DOSBox, либо в папке C:\Users\<user_name>\AppData\Local\DOSBox\dosbox-0.74.conf
- Для дальнейших изменений в конфигурации, на всякий случай, надо сделать резервную копию dosbox-0.74.conf.

Для настройки поддержки русского языка надо скопировать с заменой все файлы из архива в директорию, где хранятся конфигурационные файлы DOSBox, а после этого:

- Открыть в текстовом редакторе конфигурационный файл DOSBox.
- Найти в нем секцию [dosbox], а в ней параметр language=... .
- После знака "=" прописать путь к файлу russian.txt. Если он был скопирован в директорию DOSBox со всеми остальными файлами из архива, то измененный параметр будет выглядеть так

```
language="C:\Program Files\DOSBox-0.74\russian.txt",  
либо может быть просто - language=russian.txt
```

- Затем надо найти секцию [dos] и прописать в ней строку:

```
keyboardlayout=RU
```

После всех этих операций надо не забыть сохранить конфигурационный файл, и повторно запустить DOSBox.

Если все настройки были сделаны правильно, то уже стартовое окно DOSBox будет представлено на русском языке. Кроме этого, появилась возможность менять раскладку клавиатуры: левый ALT + правый Shift – русская раскладка, левый ALT + левый Shift – латинская.

Дополнительные настройки DOSBox

Полезной может оказаться и настройка режима оконного отображения DOSBox и параметров его видеосистемы. Один из возможных вариантов настройки секции [sdl] приведен на рис. п1.6.

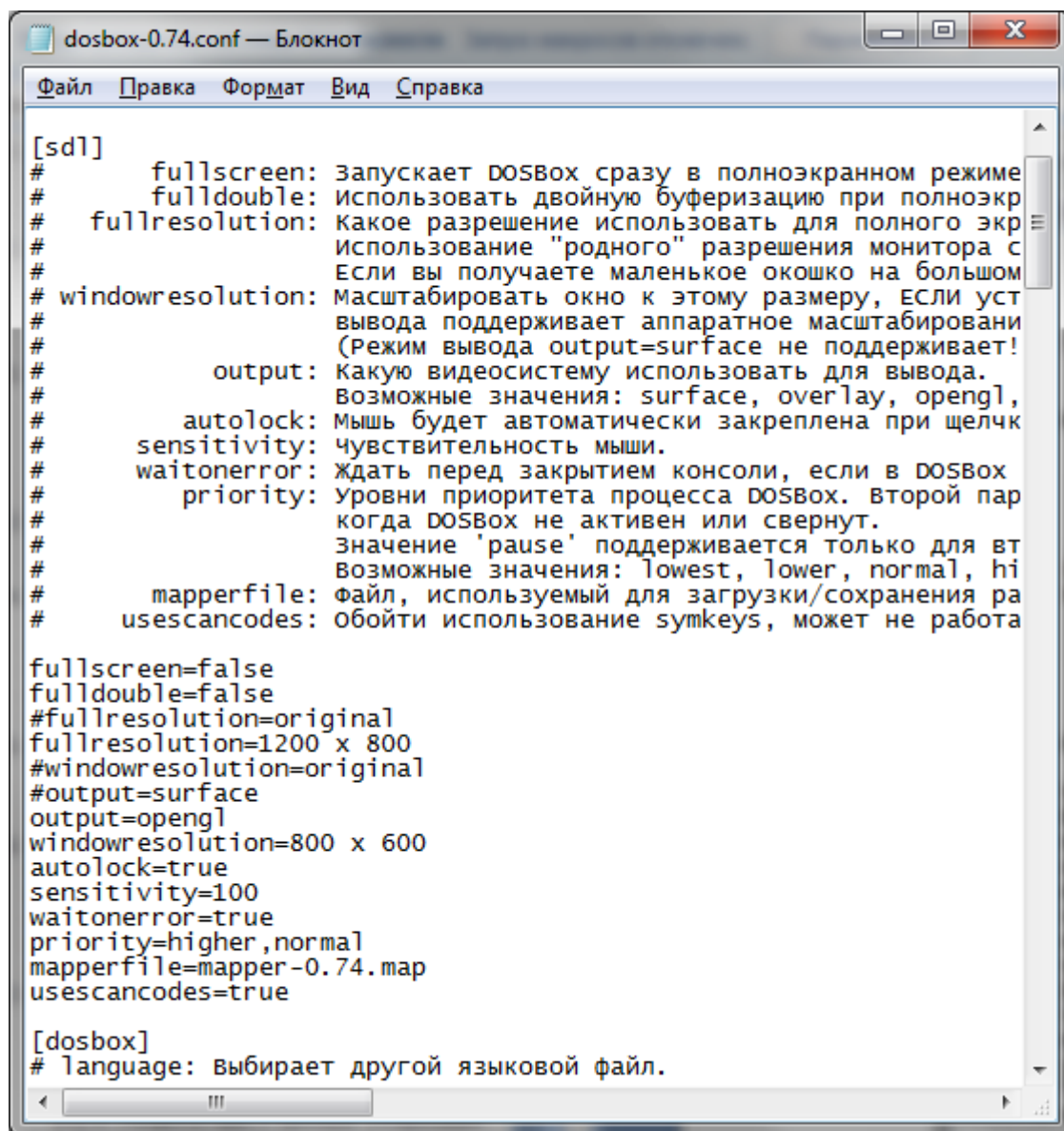


Рис. п1.6. Настройка секции [sd1] конфигурационного файла DOSBox.

Следует отметить, что в архиве русификации имеется достаточно объемный и подробный мануал по настройке DOSBox. Для запуска только одного Turbo Prolog'a может быть его подробно изучать и не следует, но я бы порекомендовал с ним познакомиться. В жизни может пригодиться, и причем с самой неожиданной стороны.

Но есть часть инструкций, которая будет полезна и при работе с Turbo Prolog. Так, при запуске программы DOSBox.exe, можно использовать параметры и опции. А это значит, что используя в качестве параметра путь к нужному нам файлу можно сразу запустить его в работу. Например, ввод в командную строку конструкции в виде:

C:\Program Files (x86)\DOSBox-0.74\DOSBox.exe" d:\TurboProlog\prolog

приведет к тому, что папку d:\TurboProlog система DOSBox автоматически смонтирует как диск C:\ и запустит файл prolog.exe. То есть, используя эту конструкцию, можно отказаться от формирования секции [autoexec] в файле конфигураций и сформировать ярлык со строкой запуска, которая бы выполняла аналогичные действия (рис. п1.7).

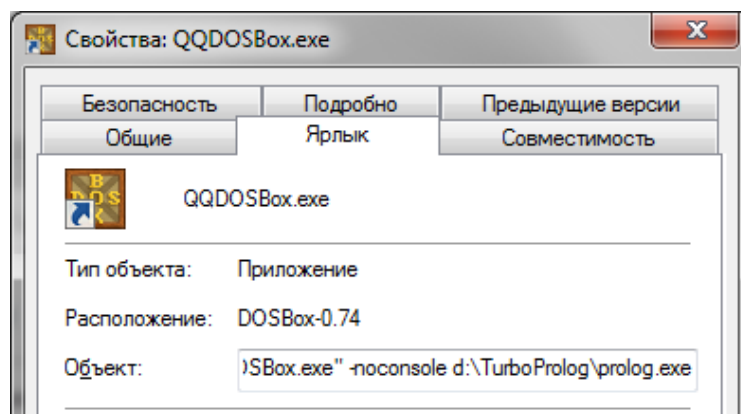


Рис. п1.7. Ярлык для запуска Turbo Prolog'a в среде DOSBox.

Кроме того, использование в команде запуска опции [-noconsole] позволит открыть DOSBox без окна служебной информации. А если еще указать опцию [-conf путь_к_конфигурационному_файлу], то появляется возможность применять свой конфигурационный файл к каждому файлу, если конечно используется много DOS-игр, а не один файл Turbo Prolog.

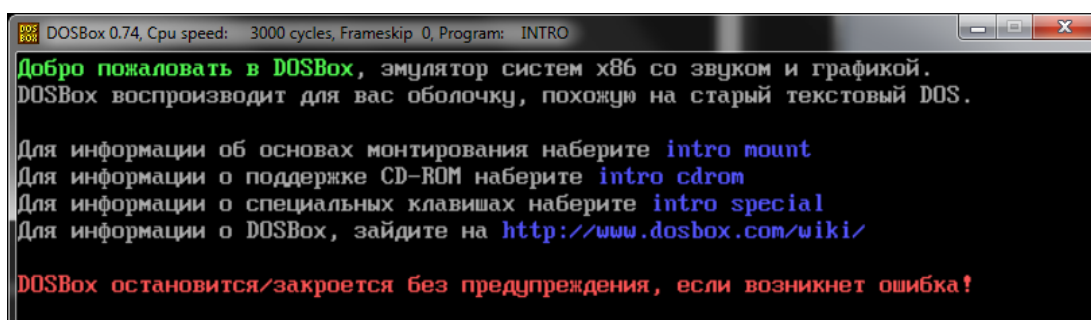


Рис. п1.7. Ярлык для запуска Turbo Prolog'a в среде DOSBox.

Помимо внешнего мануала, DOSBox имеет встроенную справку. Для ее вызова следует, находясь в среде DOSBox, ввести команду intro.

Приложение 2. Служебные предикаты Турбо-Пролога

В этом разделе будут описаны встроенные предикаты, которые включены в данную версию Турбо-Пролога. Встроенные предикаты избавляют программиста от необходимости самому определять некоторые стандартные действия. Для каждого предиката указаны типы данных и допустимые сочетания их потоковых типов:

- *i* – параметр имеет определенное значение при обращении к данному предикату
- *o* – параметр получает значение при любом обращении к предикату
- *io* – тип может быть *i* или *o* независимо от типов остальных параметров

Например, если вызываемый предикат имеет два аргумента, то возможны такие потоковые шаблоны - (*i,i*), (*o,o*), (*i,o*), (*o,i*), или более коротко - (*io,io*).

1. Стандартные предикаты ввода данных.

Турбо-Пролог предоставляет стандартные предикаты для чтения строки символов; целого, вещественного или символьного значения с клавиатуры терминала; чтения из файла. Приведенные ниже предикаты не могут непосредственно использоваться для чтения сложных объектов или списков.

Синтаксис предиката	Назначение предиката
readln (<i>СтроковаяПеременная</i>) (string) -- (o)	Читает символы из текущего входного потока, до кода "возврат каретки". Аргумент должен быть несвязанной переменной.
readint (<i>ЦелаяПеременная</i>) (integer) -- (o)	Читает цифры из текущего входного потока, до кода "возврат каретки". Аргумент должен быть несвязанной переменной.
readreal (<i>ВещественнаяПеременная</i>) (real) -- (o)	Читает вещественные символы из текущего входного потока, до кода "возврат каретки". Аргумент должен быть несвязанной переменной.
readchar (<i>СимвольнаяПеременная</i>) (char) -- (o)	Предикат readchar(...) читает один символ из текущего входного потока.
file_str (<i>ИмяФайла</i> , <i>СтроковаяПеременная</i>) File <---> String (string,string) -- (i,io)	Предикат file_str(...) читает символы из файла в строку, пока не встретится ASCII-код "конец файла".

2. Предикаты преобразования данных

char_int (*СимвольныйПараметр* , *ЦелыйПараметр*)
(char , integer) -- (i,o) (o,i) (i,i)

Предикат char_int(...) в случаях:

- (i,o) - связывает второй аргумент с десятичным ASCII-кодом первого аргумента;
- (o,i) - связывает первый аргумент с символом, имеющим ASCII-код второго аргумента;
- (i,i) - является успешным, если второй аргумент связан с ASCII-кодом первого.

str_int (*СтроковыйПараметр* , *ЦелыйПараметр*)
(string , integer) -- (i,o) (o,i) (i,i)

Предикат str_int(...) в случаях:

- (i,o) - связывает второй аргумент с двоичным эквивалентом десятичного целого, с которым связан первый аргумент;
- (o,i) - связывает первый аргумент со строкой десятичных цифр, представляющих значение, с которым связан второй аргумент;
- (i,i) - является успешным, если второй аргумент связан с двоичным эквивалентом десятичного целого, с которым связан первый аргумент.

str_char(*СтроковыйПараметр* , *СимвольныйПараметр*)

(string , char) -- (i,o) (o,i) (i,i)

Предикат str_char(...) в случаях:

- (i,o) - связывает второй аргумент с единственным символом, содержащимся в первом аргументе, который связан с этим символом;
- (o,i) - связывает первый аргумент со вторым;
- (i,i) - будет истинным, если оба аргумента связаны с одним и тем же символом.

str_real(*СтроковыйПараметр* , *ВещественныйПараметр*)
 (string , real) -- (i,o) (o,i) (i,i)

Предикат str_real(...) в случаях:

- (i,o) - связывает второй аргумент с двоичным эквивалентом десятичного вещественного числа, с которым связан первый аргумент;
- (o,i) - связывает первый аргумент со строкой десятичных цифр, представляющих значение, с которым связан второй аргумент;
- (i,i) - является успешным, если второй аргумент связан с двоичным эквивалентом десятичного вещественного, с которым связан первый аргумент.

upper_lower(*СтрокаВерхнРег* , *СтрокаНижнРег*)
 (string , string) -- (i,i) (i,o) (o,i)

Предикат upper_lower(...) в случае:

- (i,o) - связывает второй аргумент со значением первого на нижнем регистре;
- (o,i) - связывает первый аргумент со значением второго на верхнем регистре;
- (i,i) - предикат успешно выполняется, если аргументы связаны со значением одной и той же строки, но первый - на верхнем регистре, а второй - на нижнем.

3. Предикаты работы с окнами.

Синтаксис предиката	Назначение предиката
makewindow (<i>НомерОкна</i> , <i>ЦветЭкрана</i> , <i>ЦветРамки</i> , <i>ЗаголовокРамки</i> , <i>Строка</i> , <i>Столбец</i> , <i>Высота</i> , <i>Ширина</i>) (integer, integer, integer, string, integer, integer, integer, integer) -- (i,i,i,i,i,i,i,i)	Определяет область экрана как окно. Параметр <i>ЦветРамки</i> ограничивает область окна рамкой указанного цвета. Значения <i>Строка</i> и <i>Столбец</i> определяют верхнюю позицию окна, а <i>Высота</i> и <i>Ширина</i> - его размеры. Параметр <i>ЦветЭкрана</i> определяет цвет окна и символов.
shiftwindow (<i>НомерОкна</i>) (integer) -- (io)	Устанавливает или возвращает номер текущего окна.
removewindow	Удаляет текущее окно с экрана.
clearwindow	Удаляет текст из текущего окна.
window_str (<i>ЭкраннаяСтрока</i>) (string) -- (io)	- В случае (i) - связывает аргумент со строкой, выводимой в текущем окне. <i>ЭкраннаяСтрока</i> имеет то же количество линий (строк), сколько их в окне. Длина каждой строки определяется последним символом, отличным от пробела. - В случае (o) - аргумент выводится в окно. Если в аргументе больше строк, чем в окне, то выводятся только те, которые помещаются в окне и строка усекается
window_attr (<i>Атрибут</i>) (integer) -- (i)	Устанавливает значение <i>атрибута</i> для текущего окна.

4. Предикаты работы с файлами.

Турбо-Пролог определяет текущий входной данных, из которого производится чтение всех вводимых данных, и текущий выходной поток, в который производится запись данных. Обычно с текущим выходным потоком связывается дисплей, а с текущим входным потоком - клавиатура.

Однако, в ряде случаев используются операции чтения или записи данных в файлы. Для того чтобы получить доступ к файлу, он должен быть открыт. Файл может быть открыт для чтения, для записи, для добавления информации или для модификации.

Когда файл открыт, Турбо-Пролог связывает символическое (*СимволИмяФайла*) имя файла с действительным именем файла (*ИмяФайла*), используемым в каталоге ДОС. Символическое имя файла должно быть атомом и должно быть описано в программе в разделе domains как стандартный тип file. В программе тип file может встречаться в разделе domains один раз.

Синтаксис предиката	Назначение предиката
openread (<i>СимволИмяФайла</i> , <i>ИмяФайла</i>) (file , string) -- (i,i)	Открывает файл для чтения.
openwrite (<i>СимволИмяФайла</i> , <i>ИмяФайла</i>) (file , string) -- (i,i)	Открывает файл для записи. Если такой файл уже существовал в каталоге, то содержимое его уничтожается.
openappend (<i>СимволИмяФайла</i> , <i>ИмяФайла</i>) (file , string) -- (i,i)	Открывает дисковый файл для добавления информации.
openmodify (<i>СимволИмяФайла</i> , <i>ИмяФайла</i>) (file , string) -- (i,i)	Открывает файл для чтения и записи.
readdevice (<i>СимволИмяФайла</i>) (file) -- (io)	Переключает текущий входной поток данных на файл с указанным именем. Аргумент должен быть связанным и открыт для чтения, иначе происходит связывание <i>СимволИмяФайла</i> с именем активного читающего устройства.
writedevise (<i>СимволИмяФайла</i>) (file) -- (io)	Определяет текущий входной поток при условии, что указанный файл открыт либо для записи, либо для добавления информации.
closefile (<i>СимволИмяФайла</i>) (file) -- (i)	Предикат closefile(...) закрывает файл.
filepos (<i>СимволИмяФайла</i> , <i>Позиция</i> , <i>Режим</i>) (file , real , integer) -- (i,io,i)	Изменить место чтения или записи в указанном файле, который должен быть открыт для чтения и записи. Параметр <i>Режим</i> указывает позицию места чтения/записи относительно: начала файла (0), текущей позиции (1), конца файла (2)
eof (<i>СимволИмяФайла</i>) (file) -- (i)	Проверяет является ли текущая позиция в процессе чтения концом файла.
existfile (<i>ИмяФайла</i>) (string) -- (i)	Проверяет наличие файла в текущем каталоге. Используется перед попыткой открыть файл.
deletefile (<i>ИмяФайла</i>) (string) -- (i)	Уничтожает файл с именем <i>ИмяФайла</i> из каталога ДОС.
renamefile (<i>СтароеИмя</i> , <i>НовоеИмя</i>) (string , string) -- (i,i)	Переименовывает файл, если <i>НовоеИмя</i> нет в каталоге и оба имени корректны.
disk (<i>Путь_к_каталогу</i>)	Устанавливает текущее устройство и катало-

(string) -- (io)	лог.
--------------------	------

Стандартные имена устройств для **readdevice** и **writedevice** следующие: *printer*, *screen*, *keyboard*. Для записи в файл необходимо сменить текущий выходной поток данных так, чтобы им был файл, в который будут записываться данные. Позиция в файле, в которой имели место чтение или запись, может управляться предикатом *filepos*.

5. Предикаты управления экраном.

Синтаксис предиката	Назначение предиката
scr_char (<i>Строка</i> , <i>Столбец</i> , <i>Символ</i>) (integer,integer,integer) -- (i,i,io)	Выводит символ на экран с текущим атрибутом в позицию, указанную номером строки и столбца. В случае (o) - возвращает значение символа.
scr_attr (<i>Строка</i> , <i>Столбец</i> , <i>Атрибут</i>) (integer,integer,integer) -- (i,i,io)	Устанавливает или возвращает атрибут символа на экране в позиции, указанной строкой и столбцом. <i>Атрибут</i> - положительное целое число, определяющее цвет и мерцание символа
field_str (<i>Строка</i> , <i>Столбец</i> , <i>Длина</i> , <i>СтроковаяПеременная</i>) (integer,integer,integer,string) -- (i,i,i,io)	Если строка и столбец указывают на позицию внутри текущего окна, и поле данной длины, начинающееся с этой позиции, может быть помещено в это окно, то значение, с которым сцеплен аргумент <i>СтроковаяПеременная</i> , будет занесено в эту позицию, причем будет записано столько символов, сколько указано аргументом <i>Длина</i> . В случае (o) предикат возвращает значение поля.
field_attr (<i>Строка</i> , <i>Столбец</i> , <i>Длина</i> , <i>Атп</i>) (integer,integer,integer,integer) -- (i,i,i,i)	Если строка и столбец указывают на позицию внутри текущего окна, и поле данной длины, начинающееся с этой позиции, может быть помещено в это окно, то все позиции этого поля будут иметь атрибут <i>Атп</i> .
cursor (<i>Строка</i> , <i>Столбец</i>) (integer,integer) -- (i,i) (o,o)	Помещает курсор в указанную позицию текущего окна или возвращает позицию.
attribute (<i>Атрибут</i>) (integer) -- (io)	Устанавливает значение атрибута <i>Атрибут</i> по умолчанию для всех позиций экрана.

6. Предикаты обработки строк.

frontchar(*Строка* , *ПервыйСимвол* , *ОстатокСтроки*)
(string , char , string) -- (i,io,io) (o,i,i)

Предикат действует так, как если бы было определено равенство *Строка* = *ПервыйСимвол* + *ОстатокСтроки*, и либо *Строка* - связанная переменная, либо оба последних параметра - связанные переменные. В случае (i,i,i) предикат истинен, если это равенство справедливо.

fronttoken(*Строка* , *Знак* , *ОстатокСтроки*) (string , string , string) -- (i,io,io) (o,i,i)
Здесь *Строка* = *Знак* + *ОстатокСтроки* либо связанная переменная, либо оба последних параметра - связанные переменные. Предикат успешно выполняется, если второй аргумент связан с первым знаком строки, а третий аргумент - с ее остатком. Под знаком подразумевается последовательность символов, являющаяся - либо именем, соответствующим синтаксису Пролога; либо числом (предшествующий ему знак +, - и т.п. рассматривается отдельно); либо символом, но не пробелом.

frontstr(*Длина* , *ВходнаяСтрока* , *НачалоСтроки* , *ОстатокСтроки*)
(integer , string , string , string) -- (i,i,o,o)

Предикат разделяет *ВходнуюСтроку* на две части. *НачалоСтроки* будет содержать столько символов, сколько указано первым аргументом, при этом *ОстатокСтроки* будет связан с оставшимися символами.

concat(*Строка1* , *Строка2* , *Строка3*) (string , string , string) -- (i,i,o) (i,o,i) (io,i,i)
Предикат осуществляет конкатенацию первого и второго аргументов; результат связывается с третьим аргументом (*Строка3*=*Строка1*+*Строка2*). Аргументы *Строка1* и *Строка2* должны быть связанными.

str_len(*Строка*,*Длина*) (string , integer) -- (i,i),(i,o)
Предикат выполняется успешно, если в случае (i,i) первый аргумент имеет столько символов, сколько указано вторым; в случае (i,o) второй аргумент будет связан с числом символов *Строки*.

isname(*Строка*) (string) -- (i) Предикат выполняется успешно, если аргумент является именем, соответствующим синтаксису Турбо-Пролога.

7. Предикаты работы с графикой.

Графические возможности Турбо-Пролога реализованы для двух уровней: пунктирной графики, когда используются команды вида "начертить линию, проходящую через указанные точки" и графики построения "от руки", с помощью цветного пера. Перед тем как использовать команды графики Турбо-Пролога, Вы должны установить экран в графический режим, а по его окончании очистить экран и вернуться к текстовому режиму. Заметим, что текст и графика могут вместе использоваться внутри окна и на полном экране. Графический предикат имеет форму:

graphics(*Режим* , *Палитра*, *ЦветФона*) (integer , integer , integer) -- (i,i,i)

Этот предикат инициализирует экран на среднюю, высокую или сверхвысокую степень разрешения графики. Возможные значения режима и результирующего формата экрана указаны в следующей таблице.

Режим	Кол-во столбцов	Кол-во строк	Описание
1	320	200	Среднее разрешение, 4 цвета
2	640	200	Высокое разрешение, черно-белый экран
3	320	200	Среднее разрешение, 16 цветов
4	640	200	Высокое разрешение, 16 цветов
5	640	350	Сверхвысокое разрешение

Во всех случаях графика начинается с очистки экрана и установки курсора в левом верхнем углу. Цвет фона выбирается из условий:

Черный	0	Серый	8
Синий	1	Ярко-синий	9
Зеленый	2	Ярко-зеленый	10
Голубой	3	Ярко-голубой	11
Красный	4	Ярко-красный	12
Сиреневый	5	Ярко-сиреневый	13
Коричневый	6	Желтый	14
Белый	7	Ярко-белый	15

Имеются два основных графических предиката- dot u line (точка и линия).

dot(*Строка* , *Столбец* , *Цвет*) (integer,integer,integer) -- (i,i,io)

Предикат dot(...) помещает точку цвета, указанного третьим параметром, в позицию определяемую первыми двумя параметрами, которые являются целыми от 0 до 31999 и не зависят от текущего режима экрана.

line(*Строка1* , *Столбец1* , *Строка2* , *Столбец2* , *Цвет*)
(integer,integer,integer,integer,integer) -- (i,i,i,i,i)

Предикат line(...) чертит линию через точки, позиции которых указаны параметрами строки и столбца, цвет линии определяется последним аргументом.

В Турбо-Прологе есть предикаты, которые реализуют графику построения "от руки", с помощью цветного пера. Основными элементами графики этого уровня являются: изображенное на экране треугольное перо, так называемая "черепаха" и простые направляющие команды типа "вперед", "вправо" и т.п. При входе в графический режим экран очищается, появляется "черепаха", "перо" находится в хвосте. Направляется "черепаха" при помощи стандартных предикатов, а "перо" оставляет след на экране. Действие этих предикатов зависит от позиции "черепахи", направления, включено "перо" или нет, от цвета "пера". Основные предикаты черепашьей графики:

pendown - активизирует "перо"
penup - деактивизирует "перо"
pencolor(*Цвет*) (integer) -- (i) - определяет цвет следа
forward(*Шаг*) (integer) -- (i) - указывает число шагов "черепашки" из текущей позиции в текущем направлении.
back(*Шаг*) (integer) -- (i) - действует противоположным образом
right(*Угол*) (integer) -- (i) - поворачивает "черепашку" направо. Если "Угол" - связанная переменная, то "черепашка" поворачивается на указанное число градусов, если свободная, то она связывается с текущим направлением.
left(*Угол*) (integer) -- (i) - аналогично поворачивает "черепашку" налево.

8. Разные предикаты.

findall(*Переменная* , *<атом>* , *СписковаяПеременная*)

Предикат findall используется для сбора значений, полученных при возврате, в список, если атом является предикатом с аргументами, представленными действительными для Пролога именами переменных, а *Переменная* одна из переменных этого предиката, то входящая в предикат *СписковаяПеременная* будет связана со списком значений для этой переменной, которая получается, когда предикат успешно выполняется. Пример использования этого предиката:

```
domain
    name , address = string
    age = integer
    list = age*
predicates
    person( name , address , age)
    sumlist( list , age , integr )
```

```

goal
  findall( Age , Person( _,_,Age ) , L ) , sumlist ( L , Sum , N)
  Ave = Sum / N ,
  write( "\n\nЭта программа находит срений возраст лиц" ) ,
  write( "\nСредний возраст = ",Ave) , nl , nl .
clauses
  sumlist( [] , 0 , 0 ).
  sumlist( [H | T],Sum,N := sumlist( T,S1,N1),Sum=H+S1,N=1+N1.
  person("Петров","Попова, 5",22).
  person("Иванов","Институтский, 5",20)

```

random(*ВещественнаяПеременная*) (real) -- (o)

Предикат возвращает вещественное число X, удовлетворяющее условию $0 \leq X < 1$

sound(*Продолжительность, Частота*) (integer,integer) -- (i,i)

Предикат вызывает звучание сигнала нужной частоты и заданной длительности.

date(*Год,Месяц,День*) (integer,integer,integer) -- (o,o,o) (i,i,i)

В случае (o,o,o) читает дату из компьютера, а в случае (i,i,i) - устанавливает ее.

time(*Часы,Минуты,Секунды,Миллисекунды*)

(integer,integer,integer,integer)-- (o,o,o,o) (i,i,i,i)

В случае (i,i,i,i) устанавливает время в компьютере, а в случае (o,o,o,o) читает его.

system(*КоманднаяСтрока*) (string) -- (i)

Аргумент предиката может быть любой командой ДОС или выполняемым файлом.

consult(*ИмяФайла*) Загрузить базу данных из файла *ИмяФайла*.

save(*ИмяФайла*) Выгрузить базу данных в файл *ИмяФайла*.

exit Возврат к системе меню

fail Всегда имеет значение false и вызывает возврат

nl Переход на новую строку

beep Короткий звук

free(*Переменная*) Успех, если *Переменная* не связана ни с каким значением

bound(*Переменная*) Успех, если *Переменная* связана с каким-либо значением

9. Правила вычисление атрибутов экрана.

Для вычисления атрибутов экрана необходимо:

- Выбрать *один* наружный цвет и *один* цвет фона.
- Сложить указанные ниже коды этих цветов.
- Добавить 128, если нужно, чтобы объект с этим атрибутом мерцал.

Цвета фона (экрана) и их коды				Наружные цвета (символы)	
Черный	0	Серый	8	Черный	0
Синий	16	Ярко-синий	24	Синий	1
Зеленый	32	Ярко-зеленый	40	Зеленый	2
Голубой	48	Ярко-красный	72	Голубой	3
Красный	64	Ярко-	88	Красный	4
Сиреневый	80	сиреневый	104	Сиреневый	5
Коричневый	96	Желтый	120	Коричневый	6
Белый	112	Ярко-белый		Белый	7

Приложение 3. Пример программы построения дерева синтаксического анализа

В лингвистике и информатике, синтаксический анализ — это процесс сопоставления линейной последовательности лексем (слов, токенов) естественного или формального языка с его формальной грамматикой. На жаргоне программистов этот процесс известен как парсинг. Результатом этого процесса обычно является дерево разбора (синтаксическое дерево).

Синтаксический анализатор (парсер) — это программа или часть программы, выполняющая синтаксический анализ. Пример дерева разбора простейшего арифметического выражения приведен на рис. п3.1.

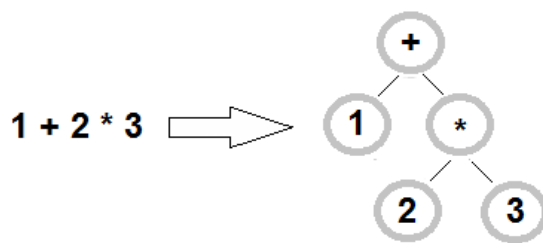


Рис. п3.1. Дерево разбора арифметического выражения.

При парсинге исходный текст преобразуется в структуру данных, обычно в дерево, которое отражает синтаксическую структуру входной последовательности и хорошо подходит для дальнейшей обработки.

Областью применения является все, что имеет синтаксис и поддается автоматическому анализу. Сюда относят языки программирования, которые ведут разбор исходного кода в процессе трансляции, SQL-запросы, структурированные данные (XML, HTML, CSS, ini-файлы), и ряд других. С точки зрения построения интеллектуальных систем этот подход наиболее важен при решении лингвистических задач понимания человеческого языка, а также в задачах машинного перевода и генерации текстов.

В качестве примера рассмотрим процесс синтаксического анализ простейших предложений русского языка. Для описания этой задачи будем использовать специальную форму записи грамматических правил.

Запись грамматических правил

Специальная форма записи грамматических правил в языке Пролог была разработана, чтобы помочь программисту в создании синтаксических анализаторов. Эта форма записи облегчает чтение программы, скрывая всю информацию, не представляющую интереса. Так как эта форма записи более краткая, чем обычная для Пролога, то вероятность сделать ошибку при написании синтаксических анализаторов значительно ниже. Любое про-

стое предложение русского языка подчиняется следующему набору строго определенных грамматических правил:

предложение —> группа_подлежащего, группа_сказуемого.
группа_подлежащего —> группа_существительного.
группа_существительного —> существительное.
группа_существительного —> прилагательное, существительное.
группа_сказуемого —> сказуемое.
группа_сказуемого —> сказуемое, глагольная_группа.
глагольная_группа —> группа_существительного.
существительное —> из_существительных.
сказуемое —> из_глаголов.
прилагательное —> из_прилагательных.
существительные —> [мама, маша, ..., ..., ...].
глаголы —> [мысль, ..., ..., ...].
прилагательные —> [милая, дорогая, ..., ..., ...].

Реализация грамматических правил на языке Prolog

На основе сформулированных выше правил грамматики, составлена Prolog-программа, код которой с комментариями приводится ниже. Пример ее работы в среде SWI-Prolog представлен на рис. п3.2. для предложений типа: "Старый папа растит дорогую машу", "Дорогая мама моет старую раму". Возможные и другие фразы на основе словаря программы.

```
% Автор: С.Хабаров
% Дата: 12.07.2013

% Группа существительного - это пара определение-
% прилагательное (если оно есть) и существительное

группа_существительного ([X|L], L, no, X) :-
    существительное (X), !.
группа_существительного ([X, Y|L], L, X, Y) :-
    прилагательное (X), существительное (Y), !.

% Предложение: набор слов есть предложение, если
% из него выделяется группа подлежащего,
% а из остатка - группа сказуемого, после чего
% остаток должен быть пустым (разбор завершен).
% Отсечение: при успехе сообщение о неудаче не выводим.
% Если отсечение не выполнено сообщаем о неудаче.

предложение (Text, Text2, semantic (StructP, StructS)) :-
    группа_подлежащего (Text, Text1, StructP),
```

```

        группа_сказуемого(Text1,Text2,StructS),
        Text2=[],
        !.
    предложение(_,_,):- writeln('Ошибка разбора\n'),fail.
% Группа подлежащего - должна начинаться с определения
% (если оно есть), затем следует существительное

    группа_подлежащего(L,L1,gr_p(X,Y)) :-
        группа_существительного(L,L1,X,Y).

% Группа сказуемого начинается с глагола и остатка в
% виде глагольной группы. Может состоять только из
% одного глагола
    группа_сказуемого([X|L],L,gr_s(X)):- сказуемое(X).
    группа_сказуемого([VERB|TOKL],TOKL1,gr_s(VERB,StuctD)):-
        сказуемое(VERB),
        глагольная_группа(TOKL,TOKL1,StuctD).

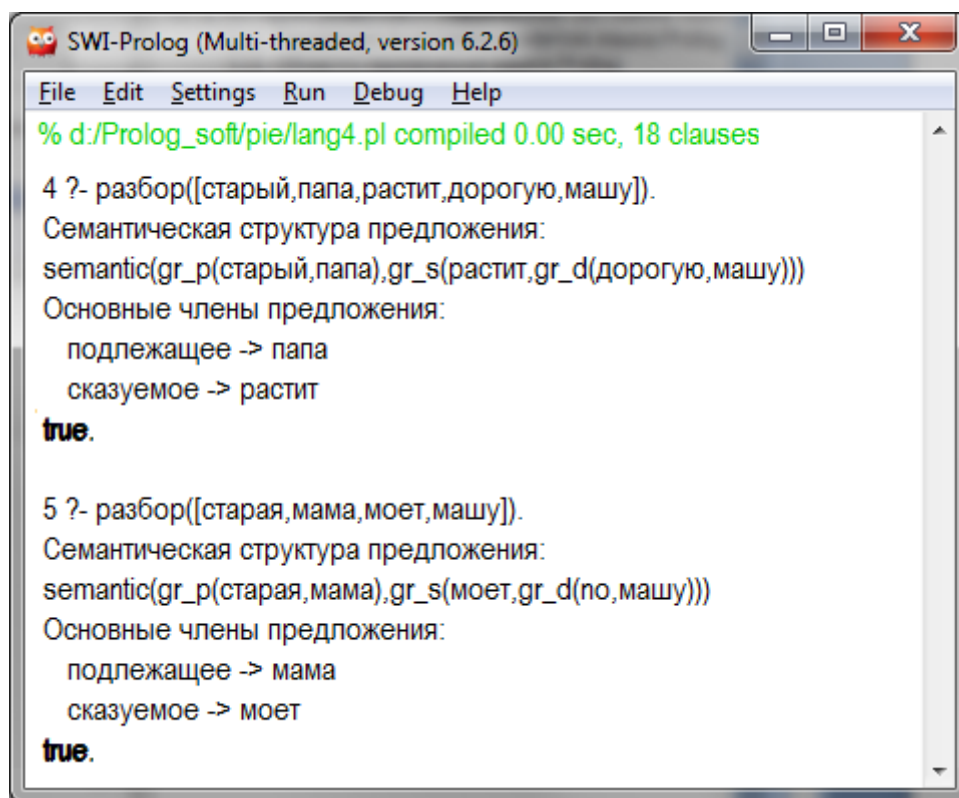
% Глагольная группа - это пара определение-прилагательное,
% выступающая как второстепенный член предложения -
% дополнение к сказуемому
    глагольная_группа(TOKL,TOKL1,gr_d(X,Y)) :-
        группа_существительного(TOKL,TOKL1,X,Y).

% Разбор предложения. Остаток после полного разбора
% должен быть равен нулю
    разбор1(Text) :- предложение(Text,End,Structure),
        End=[], writeln(Structure).
    разбор2(Text) :- предложение(Text,[],semantic(A,B)),
        writeln(A),writeln(B).
    разбор(Text) :- предложение(Text,[],semantic(A,B)),
        writeln('Семантическая структура
                предложения:'),
        writeln(semantic(A,B)),
        gr_p(_,P)=A, gr_s(S,_)=B,
        writeln('Основные члены предложения:'),
        format('    подлежащее -> ~w ~n',P),
        format('    сказуемое -> ~w ~n',S).

существительное(X) :- существительные(List),
    member(X, List).
сказуемое(X)       :- глаголы(List),
    member(X, List).
прилагательное(X)  :- прилагательные(List),
    member(X, List).

```

существительные (['мама', 'рама', 'папа', 'маша', 'машу']) .
глаголы (['моет', 'растит']) .
прилагательные (['дорогой', 'дорогая', 'дорогую', 'старый',
'старая', 'старую']) .



```
SWI-Prolog (Multi-threaded, version 6.2.6)
File Edit Settings Run Debug Help
% d:/Prolog_soft/pie/lang4.pl compiled 0.00 sec, 18 clauses
4 ?- разбор([старый,папа,растит,дорогую,машу]).
Семантическая структура предложения:
semantic(gr_p(старый,папа),gr_s(растит,gr_d(дорогую,машу)))
Основные члены предложения:
    подлежащее -> папа
    сказуемое -> растит
true.

5 ?- разбор([старая,мама,моет,машу]).
Семантическая структура предложения:
semantic(gr_p(старая,мама),gr_s(моет,gr_d(но,машу)))
Основные члены предложения:
    подлежащее -> мама
    сказуемое -> моет
true.
```

The screenshot shows the SWI-Prolog Editor window with the file `serp_semantic.pl` open. The editor has a menu bar (Файл, Править, Старт, Тест, XPCЕ, Окно, Помощь) and a toolbar. The code in the editor is as follows:

```

18
19 предложение(Text, Text2, semantic(StructF, Structs)) :-
20     группа_подлежащего(Text, Text1, StructF),
21     группа_сказуемого(Text1, Text2, Structs),
22     Text2=[],
23     !.
24 предложение(_, _, _) :- writeln('Ошибка разбора\n'), fail.
25
26 % Группа подлежащего - должна начинаться с определения
27 % (если оно есть), затем следует существительное

```

The bottom pane shows the execution results:

```

1 ?- consult('serp_semantic').
% serp_semantic compiled 0.00 sec, 18 clauses
true.

3 ?- разбор([дорогая,мама,моет,старую,рама]).
Семантическая структура предложения:
semantic(gr_p(дорогая,мама),gr_s(моет,gr_d(старую,рама)))
Основные члены предложения:
    подлежащее -> мама
    сказуемое -> моет
true.

```

The status bar at the bottom indicates: Строка: 53 Столбец: 54 Ins ANSI/Dos D:\Prolog\serp_semantic.pl Сохранен

Рис. п3.2. Примеры грамматического разбора в среде SWI-Prolog и в среде SWI-Prolog Editor.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Братко, Иван.* Алгоритмы искусственного интеллекта на языке PROLOG, 3-е издание. : Пер. с англ. — М. : Издательский дом "Вильямс", 2004. — 640 с. :
2. *Стерлинг Л., Шапиро Э.* Искусство программирования на языке Пролог. — М.: Мир, 1990.
3. Марселлус Д. Программирование экспертных систем на Турбо-Прологе — М.: Финансы и статистика, 1994
4. *Ин Ц., Соломон Д.* Использование Турбо-Пролога. — М.: Мир, 1993
5. *Шрайнер П.А.* Основы программирования на языке Пролог. Курс лекций. — ИНТУИТ, 2005. — 176 с.
6. *Цуканова Н.И., Дмитриева Т.А.* Теория и практика логического программирования на языке Visual Prolog 7. Учебное пособие для вузов. — М.: Горячая Линия – Телеком, 2013. — 232 с.
7. Язык программирования Prolog [Электронный ресурс]. — Режим доступа: открытый ресурс; постоянный адрес в Интернет: — <http://it.kgsu.ru/Prolog/oglav.html> — Загл. с экрана.
8. Visual Prolog. Техническая, справочная и обучающая информация от разработчиков (рус.) [Электронный ресурс]. — Режим доступа: открытый ресурс; постоянный адрес в Интернет: — <http://wikiru.visual-prolog.com/index.php>
9. *Costa Eduardo.* Visual Prolog для чайников [Электронный ресурс]. — Режим доступа: открытый ресурс; постоянный адрес в Интернет: http://wikiru.visual-prolog.com/index.php?title=Visual_Prolog_for_Tyros — Загл. с экрана
10. *Thomas W. De Boer.* A Beginners' Guide to Visual Prolog 7.2 [Электронный ресурс]. — Режим доступа: открытый ресурс; постоянный адрес в Интернет: — <http://download.pdc.dk/vip/72/books/deBoer/VisualPrologBeginners.pdf> — Загл. с экрана.
11. SWI-Prolog documentation [Электронный ресурс]. — Режим доступа: открытый ресурс; постоянный адрес в Интернет: — <http://www.swi-prolog.org/pldoc/index.html> — Загл. с экрана.
12. *Wielemaker Jan, Anjewierden Anjo.* Programming in XPC/Prolog [Электронный ресурс]. — Режим доступа: открытый ресурс; постоянный адрес в Интернет: <http://www.swi-prolog.org/packages/xpce/UserGuide/> — Загл. с экрана.

Хабаров Сергей Петрович

**ИНТЕЛЛЕКТУАЛЬНЫЕ
ИНФОРМАЦИОННЫЕ СИСТЕМЫ**

**PROLOG – ЯЗЫК РАЗРАБОТКИ
ИНТЕЛЛЕКТУАЛЬНЫХ И ЭКСПЕРТНЫХ СИСТЕМ**

Учебное пособие
для бакалавров и магистров направлений подготовки
230400 «Информационные системы и технологии»
и 230200 «Информационные системы»