

Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Введение в среду разработки Пролог-приложений Eclipse ProDT

Методические указания к лабораторным работам по дисциплине
“Методы и системы искусственного интеллекта”
для студентов дневной и заочной форм обучения
направления 09.03.02 — “Информационные системы и технологии”

**Севастополь
2015**

УДК 004.42 (075.8)

Введение в среду разработки Пролог-приложений Eclipse ProDT: методические указания к лабораторным работам по дисциплине “Методы и системы искусственного интеллекта” для студентов дневной и заочной формы обучения направления 09.03.02 — “Информационные системы и технологии” / СевГУ ; сост. **В. Н. Бондарев**. — Севастополь: Изд-во СевГУ, 2015. — 28 с.

Цель указаний: оказать методическую помощь студентам дневной и заочной форм обучения направления 09.03.02 — “Информационные системы и технологии” в освоении в интегрированной среды разработки Eclipse ProDT, используемой при выполнении лабораторных работ по дисциплине “Методы и системы искусственного интеллекта”

Методические указания составлены в соответствии с требованиями программы дисциплины “Методы и системы искусственного интеллекта” для студентов дневной и заочной форм обучения направления 09.03.02 — “Информационные системы и технологии” и утверждены на заседании кафедры Информационных систем (протокол № от 2015 г.)

Допущено учебно-методическим центром СевГУ в качестве методических указаний.

Рецензент: Кожаяев Е.А., к.т.н., доцент кафедры информационных технологий и компьютерных систем

СОДЕРЖАНИЕ

Введение.....	4
1. Платформа разработки приложений Eclipse.....	5
2. Установка среды Eclipse и ProDT плагина... ..	6
2.1. Установка исполняющей среды Java и Eclipse Classic IDE	6
2.2. Установка ProDT плагина и среды SWI-Prolog	6
3. Практическая работа в среде Eclipse ProDT	6
3.1. Настройка среды ECLIPSE ProDT	6
3.2. Проекция Prolog	8
3.3. Введение в Пролог	9
3.4. Простейший проект.....	12
3.5. Интерпретатор SWI-Prolog.....	17
3.5.1. Помощь	17
3.5.2. История команд	18
3.5.3. Отладка программы.....	19
Библиографический список.....	23
Приложение А. Пример проекта “Поиск в ширину”.....	24
Приложение Б. Примеры операций ввода-вывода	26

ВВЕДЕНИЕ

ProDT — это среда для разработки Пролог-приложений. Язык Пролог был разработан в начале семидесятых годов 20-го века. Само название языка определяет его суть — *язык логического программирования*.

Пролог — язык, предназначенный для программирования самых различных приложений, прежде всего тех, которые базируются на использовании моделей представления знаний и логического вывода [1]. Диапазон применения языка достаточно широк: реляционные базы данных; экспертные системы; понимание естественного языка; математическая логика; перевод с одного языка на другой; символьные вычисления, web-приложения и др.

Пролог существенно отличается от традиционных языков программирования. Программа на Прологе в большей степени представляет собой декларативное описание отношений некоторой предметной области. Выполнение программы заключается в постановке вопросов, относящихся к предметной области, и автоматическом поиске ответов на вопросы с помощью встроенных механизмов логического вывода.

Существуют различные версии языка Пролог: Quintus-Prolog, Micro-Prolog, Arity Prolog, Visual Prolog, SWI-Prolog и др. Первая эффективная реализация Пролога была создана в Эдинбургском университете в 1977. Синтаксис данной версии стал фактическим стандартом. В методических указаниях рассматривается свободно распространяемый SWI-Prolog, который поддерживает работу как в указанном стандарте синтаксиса, так и в стандарте ISO. **SWI-Prolog** содержит весь арсенал средств, необходимых для разработки больших программных систем. Он также поддерживает взаимодействие с программами, написанными на языках C/C++, имеет интерфейсы для работы с базами данных, различными web-инструментами (HTTP, SGML/XML, RDF, Semantec Web, SSL).

Рассматриваемая в методических указаниях среда **ProDT**, обеспечивающая поддержку разработки приложений с помощью **SWI-Prolog** (XSB и B Prolog), представляет расширение (дополнение) для платформы **Eclipse**. Программирование задач искусственного интеллекта на языке Пролог в современной интегрированной среде поддержки разработок **Eclipse** должно обогатить студентов как освоением новых для них принципов *логического программирования*, свойственных Прологу, так и знакомством с самой средой, которая обеспечивает интеграцию самых различных инструментов.

Основная цель методических указаний оказать помощь студентам в освоении интегрированной среды разработки **Eclipse ProDT**, используемой при выполнении лабораторных работ по дисциплине «Методы и системы искусственного интеллекта». Данные методические указания являются логическим продолжением методических указаний [2,3].

1. ПЛАТФОРМА РАЗРАБОТКИ ПРИЛОЖЕНИЙ ECLIPSE

Eclipse ([i'klips] от англ. *затмение*) представляет собой основанную на **Java** интегрированную инструментальную платформу с открытым исходным кодом (OpenSource), предназначенную для разработки кроссплатформенных приложений. По сути, платформа Eclipse — это расширяемая среда разработки и набор сервисов для создания **программного обеспечения (ПО)** на основе встраиваемых компонентов (**плагинов**). Eclipse также включает в себя среду разработки плагинов (**PDE**), что дает разработчикам инструментарию возможность предложить свои расширения к Eclipse.

Изначально проект разрабатывался в **IBM** как корпоративный стандарт интегрированной среды разработки (**IDE**) для различных платформ. С 2004г. проект разрабатывается и поддерживается независимой некоммерческой организацией **Eclipse Foundation** (www.eclipse.org). Среди участников Eclipse Foundation следует назвать компании IBM, Nokia, Oracle, SAP, OBEO, Ericsson, Intel и др.

Хотя платформа Eclipse и написана на Java, её возможности не ограничиваются разработками на этом языке. Помимо Java, в Eclipse имеются расширения, поддерживающие разработку ПО на языках **C/C++**, **FORTRAN**, **COBOL**, **PHP**, **Perl**, **Python**, **Groovy**, **Erlang**, **Ruby** и др. Множество расширений дополняет Eclipse средствами для работы с базами данных, моделирования, разработки графических приложений, серверов приложений и др. В настоящее время имеется более 1000 плагинов, расширяющих возможности Eclipse (<http://marketplace.eclipse.org/>).

В силу бесплатности и промышленного качества Eclipse постепенно завоевывает позиции корпоративного стандарта для разработки самых различных приложений во многих организациях.

Последние версии Eclipse: Eclipse 3.3.2 (**Europa**), Eclipse 3.4 (**Ganymede**, 2008 г.), Eclipse 3.5 (**Galileo**, 2009г.), Eclipse 3.6 (**Helios**, 2010г.), Eclipse 3.7 (**Indigo**, 2011г.).

Выход следующей версии Eclipse 4.2 (**Juno**) запланирован на июнь 2012г. В рамках проекта Juno разрабатывается более 50 подпроектов.

Подпроекты Eclipse сгруппированы в пакеты в соответствии с основными потребностями разработчиков и размещены по адресу: <http://www.eclipse.org/downloads/>. Более детальные сведения о проекте Eclipse и его подпроектах, а также сведения по архитектуре Eclipse приведены в [2,3,4,5].

В настоящих методических указаниях рассматривается только инструментарий разработки Пролог-приложений **Eclipse ProDT** и краткое введение в систему **SWI-Prolog** [6].

2. УСТАНОВКА СРЕДЫ ECLIPSE И ProDT ПЛАГИНА

2.1. Установка исполняющей среды Java и Eclipse Classic IDE

Порядок установки исполняющей среды **JRE** (Java Runtime Environment) и интегрированной среды разработки приложений **Eclipse Classic IDE** подробно описан в [3]. Ниже предполагается, что указанные установки уже выполнены и все файлы среды **Eclipse Classic IDE** находятся в папке **C:\Eclipse**.

2.2. Установка ProDT плагина и среды SWI-Prolog

Для установки **ProDT** необходимо выполнить следующее:

- 1) загрузить из Интернета архив с **ProDT** плагином **ar.com.tadp.prolog-1.0.0.zip** (15,8 Мбайт), который находится по адресу [http://sourceforge.net/projects/prodevtools/files/Prolog Development Tools/1.0.0/](http://sourceforge.net/projects/prodevtools/files/Prolog%20Development%20Tools/1.0.0/);
- 2) разархивировать указанный архив в папку **c:\eclipse\dropins**:

```
c:\eclipse
|->dropins
    |->ar.com.tadp.prolog-1.0.0
    |->dltk-core-sdk-R-2.0-201006161315
    |->emf-runtime-2.5.0
```

- 3) перейти на страницу <http://www.swi-prolog.org/download/stable>, выбрать ссылку **SWI-Prolog/XPCE 5.10.5 for Windows NT/2000/XP/Vista/7** и загрузить файл **w32pl5105.exe** (8.7 Mb) — установочный исполняемый файл SWI-Prolog для ОС Windows;

- 4) дважды щелкнуть мышью на файле **w32pl5105.exe** и выполнить типовую установку SWI-Prolog в папку, предлагаемую по умолчанию в процессе установки — **C:\Program Files\pl** (в процессе установки требуются библиотеки времени выполнения Visual C++, если на вашем компьютере они отсутствуют, то откройте страницу **Microsoft Visual C++ 2008 SP1 Redistributable Package (x86)**¹, загрузите исполняемый файл **vcredist_x86.exe** и выполните его).

Среда **Eclipse** с установленным плагином **ProDT** готова для запуска.

3. ПРАКТИЧЕСКАЯ РАБОТА В СРЕДЕ ECLIPSE ProDT

3.1. Настройка среды ECLIPSE ProDT

Основные понятия интерфейса пользователя среды Eclipse подробно описаны в [2,3]. Там же описано как выполнить первый запуск Eclipse.

После запуска среды Eclipse с установленным плагином ProDT требуется выполнить несложную настройку. Для этого выберите пункт меню **Окно > Па-**

¹ <http://www.microsoft.com/downloads/details.aspx?displaylang=en&FamilyID=a5c84275-3b97-4ab7-a40d-3802b2af5fc2>

параметры и откройте страницу **Prolog > Interpreters** (рис. 3.1). Нажмите на указанной странице кнопку **Add** и заполните открывшееся окно **Add Interpreter** в соответствии с рисунком 3.2. В поле **Interpreters executable** укажите путь **C:\Program Files\pl\bin\swipl.exe** к исполняемому файлу, обеспечивающему запуск среды SWI-Prolog в режиме консоли, и нажмите кнопку **ОК**. Закройте окно **Параметры**, нажав кнопку **ОК**.

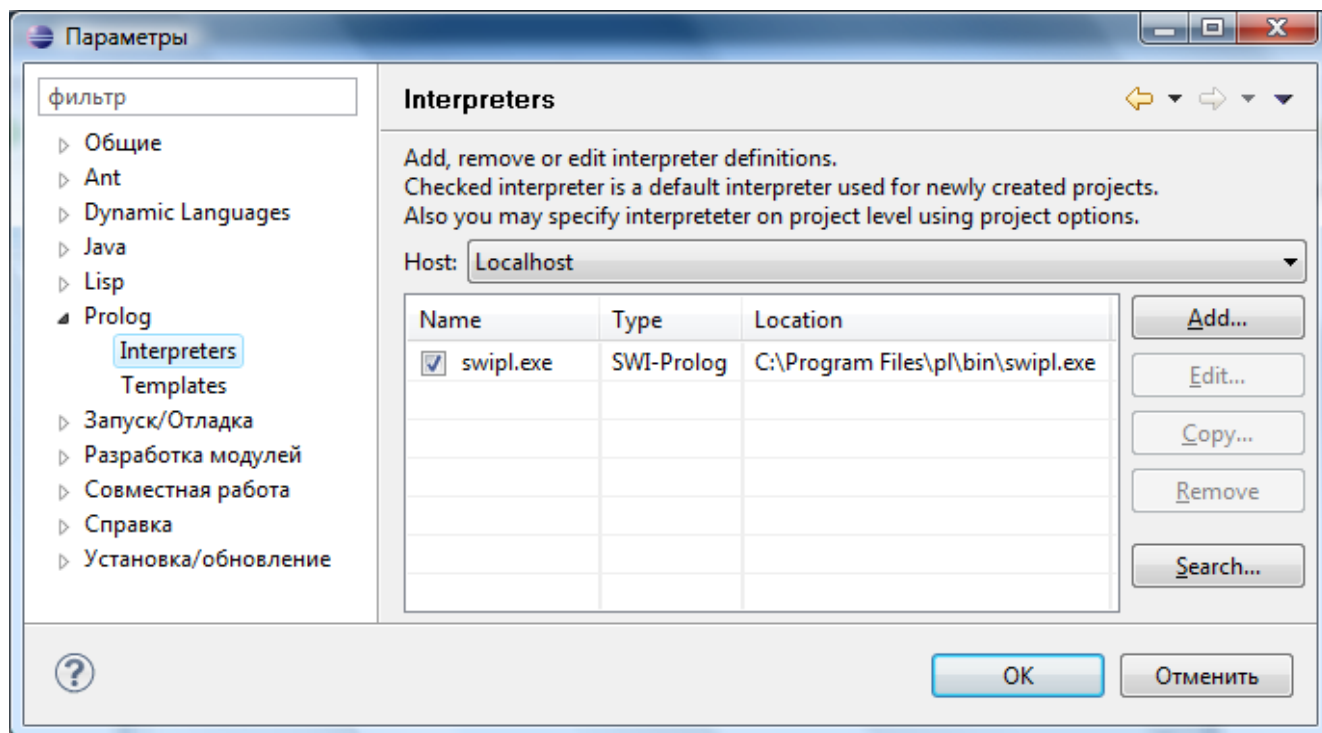


Рисунок 3.1 — Страница **Prolog > Interpreters**

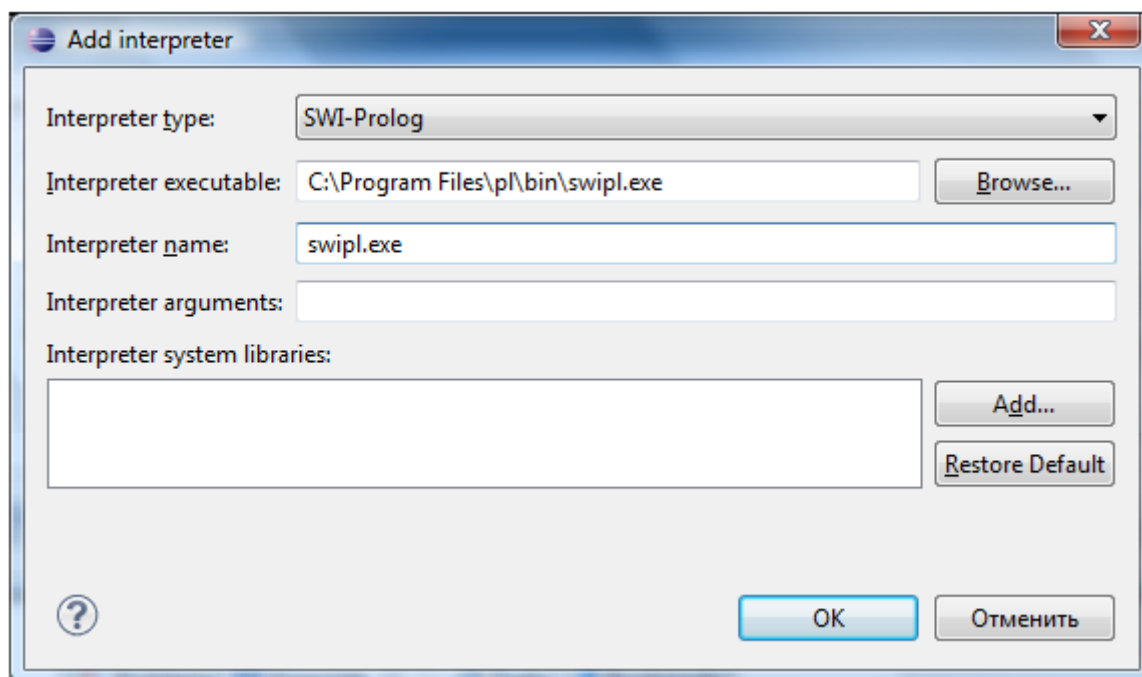


Рисунок 3.2 — Окно добавления пролог-интерпретатора

3.2. Проекция Prolog

На рис. 3.3. изображена проекция Prolog с открытым проектом. В этой проекции рабочего стола можно выделить несколько основных составляющих интерфейса пользователя, которые на рисунке заключены в прямоугольники (окна) и обозначены цифрами.

Прежде всего, это основные *элементы управления средой*:

- **Строка меню** (окно 1) — содержит пункты меню платформы Eclipse с набором функций для работы с проектами;
- **Панель инструментов** (окно 2) — содержит набор кнопок, которые обеспечивают быстрый выбор того или иного инструмента;
- **Открытие проекции** (окно 3) — кнопка, позволяющая выбрать необходимую проекцию (перспективу) из списка имеющихся проекций;
- **Текущая проекция** (окно 4) — отображает имя текущей активной проекции;
- **Показать панель как быструю панель** (окно 9) — кнопка, которая позволяет осуществить быстрое открытие одной из панелей (представлений), указанных ниже. В дальнейшем будем называть эту кнопку просто «**Быстрая панель**».

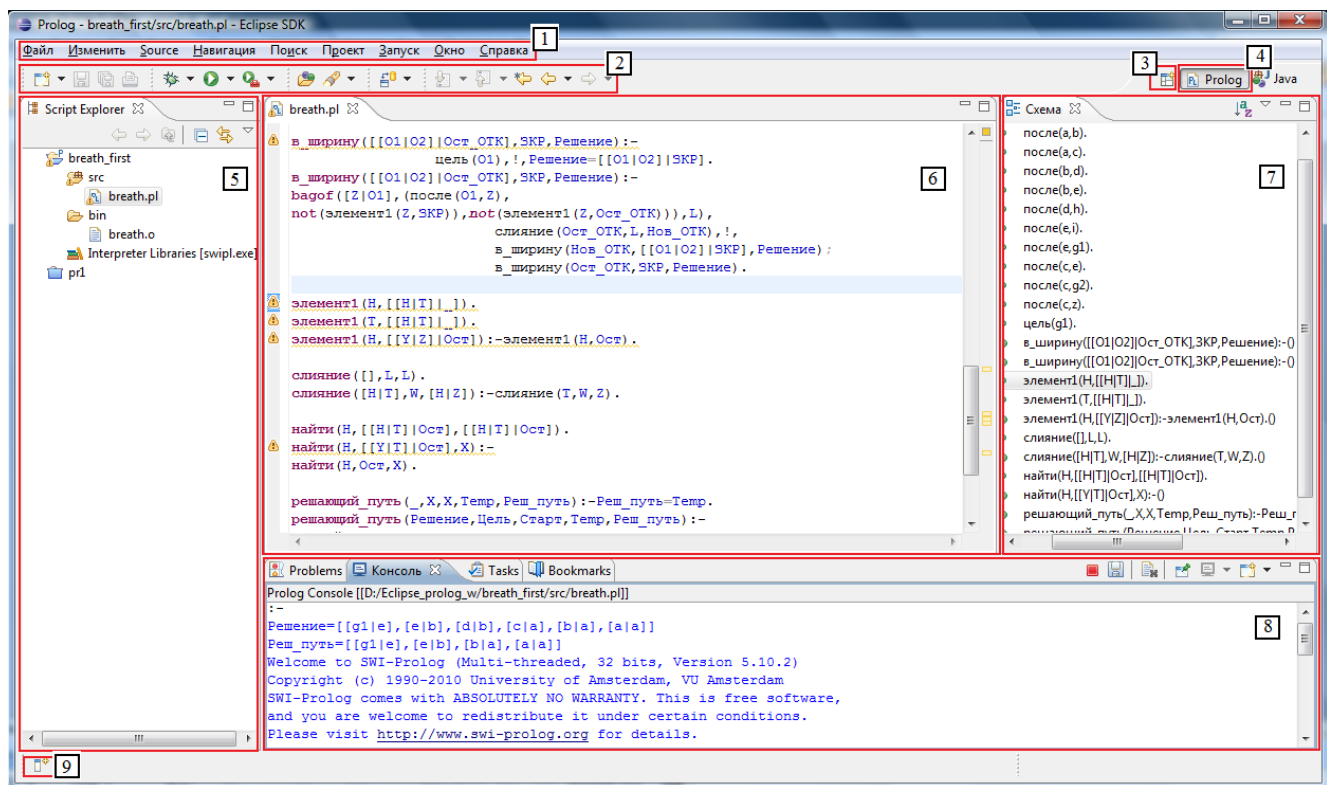


Рисунок 3.3 — Проекция Prolog

В окнах, обозначенных номерами 5,6,7 и 8 (рис. 3.3), отображаются различные *панели* (представления), расположение которых, при необходимости, можно менять:

- **Навигатор** (Script Explorer) (окно 5) — отображает структуру рабочей области в виде каталогов и файлов, входящих в проект;

- **Редактор** (Editor) (окно 6) — обеспечивает ввод и редактирование файлов проекта;
- **Схема** (Outline) (окно 7) — обеспечивает отображения структуры файла, который в данный момент открыт в окне редактора;
- **Неполадки** (Problems) (окно 8) — отображает предупреждения и ошибки компиляции;
- **Консоль** (Console) (окно 8) — системная консоль, используемая для ввода-вывода данных программы, а также для ввода команд SWI-Prolog;
- **Задачи** (Tasks) (окно 8) — отображает список задач, которые вы запланировали;
- **Закладки** (Bookmarks) (окно 8) — используется для быстрого перехода к закладке, установленной в окне редактора.

Кнопка **«Быстрая панель»**, находящаяся в левом нижнем углу рабочего стола, открывает список доступных панелей и добавляет выбранную панель поверх окна 5 (рис 3.3). При этом вновь открытая панель закрывает собой некоторую часть рабочего стола. Имеется возможность добавить открываемую панель в любое из окон 5,6 или 7. Для этого необходимо нажать мышью на закладке заголовка панели и перетащить её в необходимое окно. В процессе перетаскивания панели на рабочем столе будет появляться прямоугольная рамка, показывающая новое расположение панели.

3.3. Введение в Пролог

Программа на Прологе представляет совокупность *утверждений* предметной области, записанных с использованием предикатов. Выполнение программы сводится к доказательству (логическому выводу) целевого утверждения. Механизм логического вывода, базирующийся на принципе резолюции, встроен в Пролог-систему [1].

В математике *предикатом* называется неоднородная двузначная логическая функция от любого числа аргументов. Ее записывают в виде $P(x_1, x_2, \dots, x_n)$ и называют *n-местным предикатом*. Здесь аргументы x_1, x_2, \dots, x_n — принадлежат одному и тому же или различным множествам, представляющим объекты предметной области. Аргументы x_1, x_2, \dots, x_n называют *предметными переменными*, а их конкретные значения — *предметными постоянными*. Функциональную букву P называют предикатной буквой.

В общем случае *n-местный предикат* $P(x_1, x_2, \dots, x_n)$ задает отношение между аргументами x_1, x_2, \dots, x_n , которое означает, что “ x_1, x_2, \dots, x_n находятся между собой в отношении P ”. Например, полагая, что предикатная буква P представляет отношение «произведение», можно трехместному предикату $P(x_1, x_2, x_3)$ дать следующую интерпретацию: “ x_1 есть произведение x_2 на x_3 ”.

В языке Пролог при записи утверждений вместо предикатных букв используют имена, которые записывают строчными буквами. **В конце утверждения ставится точка.**

Существуют два типа утверждений Пролога: *факты и правила*.

Факт представляет собой истинное утверждение. Факт на Прологе представляется в виде предиката. Например, утверждение “Игорь — отец Святослава” записывается на Прологе в следующей форме [1]:

отец('Игорь', 'Святослав').

Здесь **отец** — это имя предиката, а аргументы **'Игорь'** и **'Святослав'** представляют собой символьные константы. *Символьные константы* должны начинаться со строчной буквы или заключаться в одинарные кавычки.

В Прологе используется соглашение, позволяющее отличать имена конкретных объектов (констант) от имен переменных. Имя, начинающееся с прописной буквы или символа подчеркивания, рассматривается как *переменная*. Так, **X, Y, T34, _a, _132** — это имена переменных.

Правило — представляет собой утверждение, которое истинно при определенных условиях. Например, утверждение “X — дед Y” верно при условии, что X — это отец Z, где Z — это один из родителей Y. На Прологе это правило запишется в виде

дед(X,Y): – отец (X,Z), родитель(Z,Y).

Правило состоит из *заголовка (головы)* и *тела*. Заголовок и тело соединяются знаком “:-”, который соответствует импликации “ \leftarrow ”. Запятая, записанная в теле правила, соответствует логической связке “и”. Приведенное выше правило интерпретируется следующим образом: “отношение **дед(X,Y)** верно, если верны отношения **отец(X,Z)** и **родитель(Z,Y)**”. Заголовок правила представляет факт, для определения которого собственно и предназначено правило. В рассматриваемом случае — это **дед(X,Y)**. Данный факт будет иметь место, если будут верными утверждения, образующие тело правила, которые называют *условиями*.

В теле правила можно использовать логическую связку “или”, которую обозначают точкой с запятой. Например,

родитель(X,Y): – отец(X,Y); мать(X,Y).

Иными словами, X является родителем Y, если X — это отец или мать Y.

В Прологе используют однострочные и многострочные комментарии. Однострочный комментарий начинается со знака “%”, а многострочный ограничивается символами: “/*...*/”.

Совокупность фактов и правил в Прологе образуют *базу данных*. Рассмотрим пример базы данных, содержащей отношения родства:

% Факты

отец('Иван','Сергей').

% Иван — отец Сергея

отец('Иван','Анна').

% Иван — отец Анны

мама('Мария','Сергей').

% Мария — мать Сергея

% Правило

/ О и М являются родителями ребенка Р,*

*если О является отцом Р, а М является мамой Р */*

родители(О,М,Р): – отец(О,Р), мама(М,Р).

Когда база данных создана и загружена в память Пролог-системы, к ней можно обращаться с вопросами (запросами). Например, можно задать вопрос: “Является ли Иван отцом Сергея?” Для этого необходимо в окне консоли Пролог-системы ввести *целевое утверждение*:

?– отец('Иван','Сергей').

Обращение с вопросом к Пролог-системе инициирует процесс поиска (доказательства) соответствующих утверждений в базе данных. Система начинает искать факты, *сопоставимые* с фактом целевого утверждения. При этом выполняется *процедура унификации*. Два факта сопоставимы, если совпадают имена предикатов, и соответствующие аргументы попарно сопоставимы. В рассматриваемом случае Пролог находит в базе данных факт **отец('Иван','Сергей')**, сопоставимый с целевым утверждением. Обнаружив этот факт, Пролог-система ответит **Yes** (да).

К рассмотренной базе данных можно обращаться и с другими вопросами. Например, ответом на вопрос

? – отец('Сергей','Иван').

будет **No** (нет), так как в базе данных нет факта, сопоставимого с вопросом. На вопрос

? – отец('Иван',P). % Кто дети Ивана?

Пролог-система выдаст ответ **P='Сергей'**. Если ввести с клавиатуры “;”, то Пролог-система продолжит поиск, и получит второе значение **P='Анна'**. Если еще раз ввести точку с запятой, то система вернет значение **No**. Можно задать и более сложный вопрос: “Кто родители Сергея?”:

? – родители(O,M,'Сергей').

В результате получим

**O='Иван';
M='Мария';
No.**

В данном случае Пролог-система воспользовалась правилом и попытку доказательства целевого утверждения

родители(O,M,'Сергей')


заменила на доказательство двух новых подцелей **отец(O,'Сергей')** и **мама(M,'Сергей')**. В результате было установлено, что первая подцель достижима, если **O='Иван'**, вторая подцель достижима, если **M='Мария'**. Следовательно, целевое утверждение верно при указанных значениях переменных.

Даже на этом простом примере, содержащем несколько строк кода, видна сила языка Пролог. Чтобы ощутить это представьте, сколько строк кода вам пришлось бы написать на алгоритмическом языке для ответов на приведенные выше вопросы. И обратите внимание, что в случае с Прологом мы не написали ни строчки кода, чтобы обрабатывать запросы к базе данных. Механизм поиска ответов на запросы (принцип резолюции) уже реализован в самой Пролог-системе.

3.4. Простейший проект

Приступим к созданию и выполнению приведенной выше программы с использованием Eclipse ProDT. Для этого необходимо выполнить несколько шагов.

Шаг1. Создание проекта

Выберите **Файл > Создать > Проект** или нажмите на панели инструментов стрелку рядом с кнопкой  — **Создать** и в выпадающем списке выбрать **Проект**. В открывшемся окне мастера проектов следует выбрать **Prolog > Prolog Project** и нажать кнопку **Далее**. В результате откроется окно создания нового проекта (рис. 3.4).

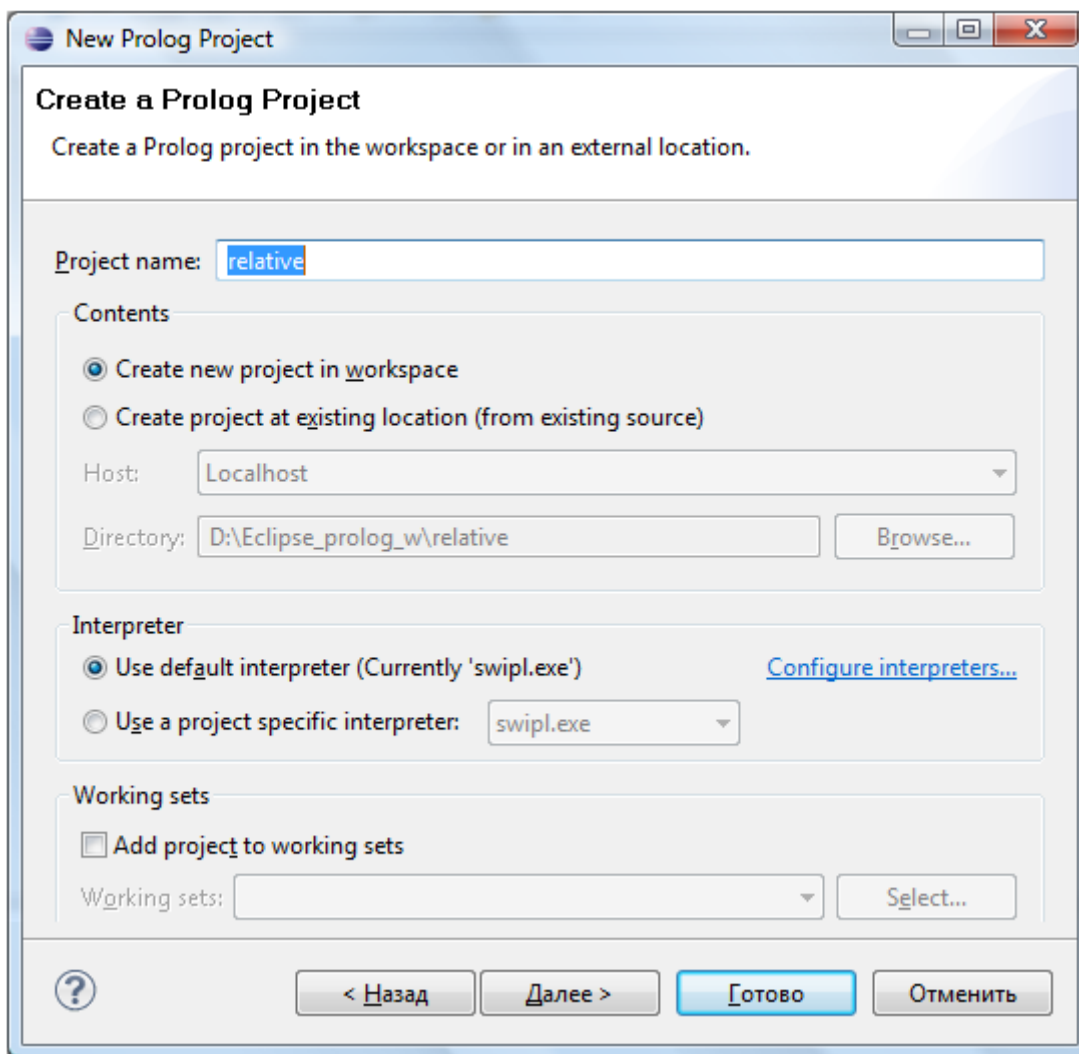



Рисунок 3.4 — Менеджер создания проекта

В окне следует ввести имя проекта, например **relative**, и нажать кнопку **Готово**.

В результате будет создан новый проект. При этом в рабочей области создаётся папка проекта **relative**, которая появится на панели **Навигатор (Script**

Explorer). Папка будет содержать пустую папку **src**, которая предназначена для хранения исходных файлов проекта.

Шаг2. Ввод и редактирование программы

Щелкните левой клавишей мыши на папке **src**, а затем выберите **Файл > Создать > Файл** или нажмите на панели инструментов стрелку рядом с кнопкой  — **Создать** и в выпадающем списке щелкните мышью на строке **Файл**. Откроется окно создания файла, в поле **Имя файла** введите имя и тип файла, в котором будет храниться исходный код программы, например **parents.pl** и нажмите кнопку **Готово**. Файлы с исходными кодами Пролог программ должны иметь тип **.pl** (либо **.pro**).

В файл **parents.pl**, открытый в окне редактора, введите текст рассмотренной выше программы. Результаты ввода изображены на рис. 3.5. При вводе программы символьные константы с целью упрощения записывались со строчной литеры. Поэтому они не заключены в кавычки.

Сохраните файл, нажав **Ctrl+S** (или выбрав **Файл > Сохранить** или щелкнув правой кнопкой мыши на панели с открытым файлом и выбрав пункт меню **Сохранить**).

Обратите внимание, что редактор показывает парные скобки и выделяет цветом имена предикатов, констант, переменных и комментариев.

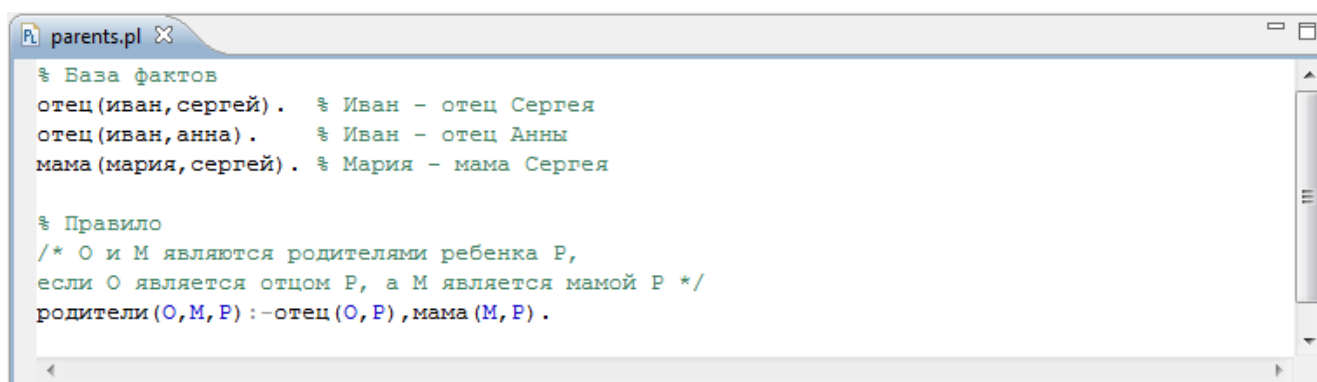


Рисунок 3.5 — Окно редактора с Пролог программой

Редактор позволяет выполнять автоматическое завершение вводимых утверждений, если они ранее уже вводились или представляют имена встроенных объектов известных Пролог-системе. Например, напечатав литеры “ро” и нажав **Ctrl+Space**, получим список утверждений, которые начинаются этими литерами (рис.3.6), если таких утверждений несколько. Если утверждение только одно, то сразу выполняется вставка его в текст программы. При этом в отдельном окне демонстрируется комментарий, предшествующий выбранному утверждению. Если выбрать утверждение из появившегося списка двойным щелчком мыши, то будет выполнена его вставка в программу.

Редактор также позволяет осуществлять быструю навигацию по утверждениям программы. Для этого нажмите клавиши **Ctrl+O** и в открывшемся окне начните печатать имя утверждения. Это позволит быстро перейти к требуемому утверждению.

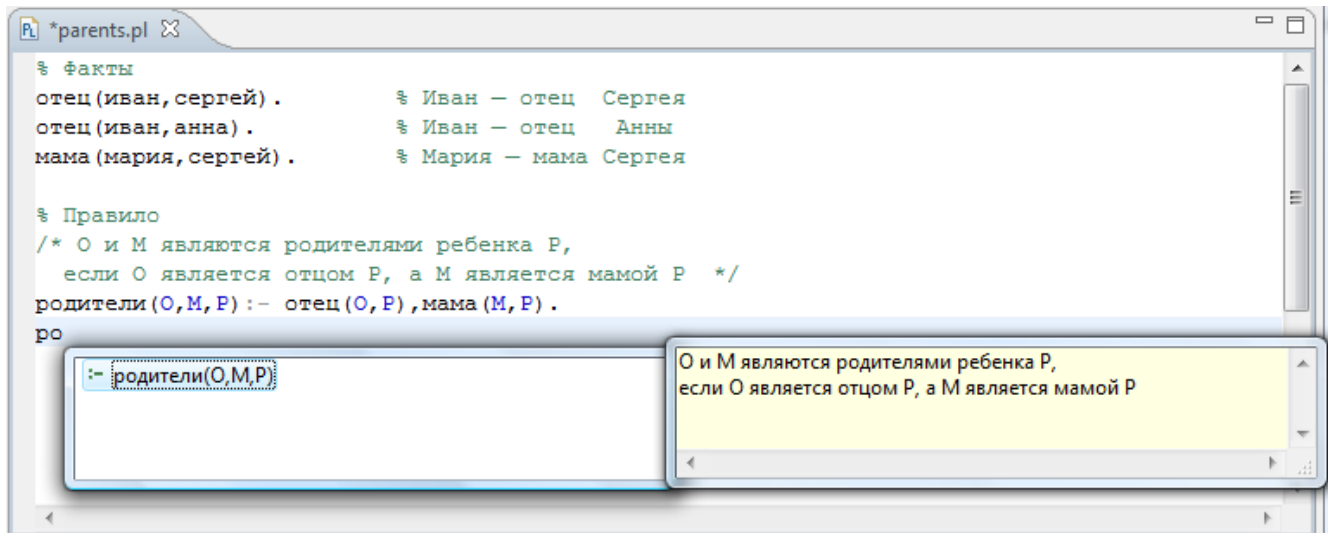


Рисунок 3.6 — Автозавершение утверждений

Шаг3. Компиляция программы

Компиляция программы выполняется автоматически при сохранении исходного файла. Если в программе будут обнаружены синтаксические ошибки, то соответствующие строки программы в окне редактора отмечаются метками в виде красных кружочков с крестиками, а сами сообщения об ошибках выводятся на панель **Неполадки (Problems)**.

На рис. 3.7 изображен пример программы с ошибками. Здесь при вводе факта **отец(иван,анна)** была напечатана лишняя скобка, а при вводе факта **мама(мария,сергей)** вместо запятой была напечатана точка.

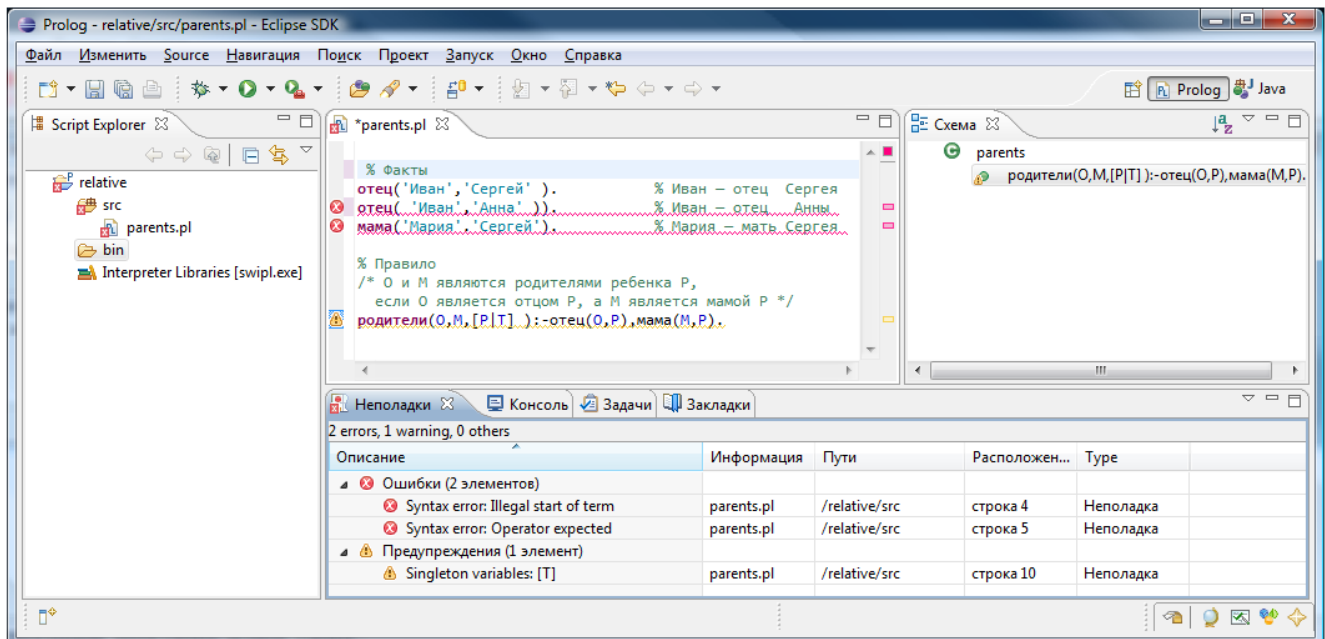


Рисунок 3.7 — Пример программы с синтаксическими ошибками

Метка в виде желтого треугольника, изображенная на рис. 3.6, обозначает предупреждение. Предупреждения не являются ошибками, однако их следует

анализировать, т.к. они могут приводить к ошибкам во время выполнения программы.

Шаг 4. Вызов интерпретатора

Чтобы доказать (выполнить) то или иное целевое утверждение следует запустить SWI-интерпретатор. При запуске интерпретатора программа повторно компилируется и её утверждения помещаются в базу данных Пролог-системы. Для запуска SWI-интерпретатора щелкните правой кнопкой мыши в окне редактора, в появившемся списке выберите **ProDT > Consult in Console**. В окне консоли откроется сессия работы с интерпретатором SWI-Prolog и будут выведены сообщения, изображенные на рис. 3.8.

Если в процессе компиляции утверждений программы будут обнаружены синтаксические ошибки, то выводятся соответствующие сообщения, но обработка оставшихся утверждений программы продолжается. При этом утверждения, не содержащие ошибок, компилируются и добавляются в базу данных Пролог-системы.

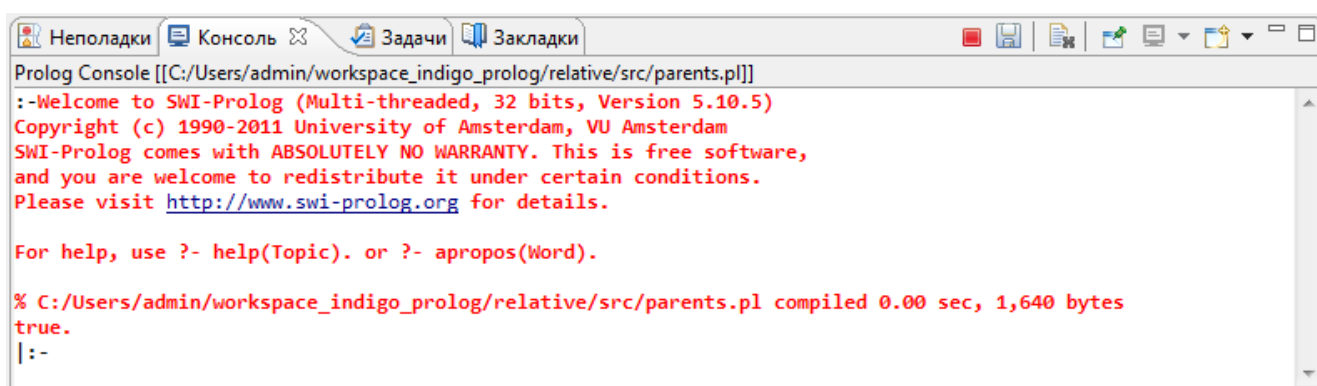


Рисунок 3.8 — Сообщения в окне консоли

Теперь в окне консоли можно вводить команды SWI-Prolog или целевые утверждения (запросы). Они будут интерпретироваться и выполняться. Например, чтобы просмотреть список утверждений, содержащихся в базе данных, введите в окне консоли слово **listing**, поставьте в конце точку и нажмите **Enter**. На экране появится список утверждений базы данных (рис. 3.9).

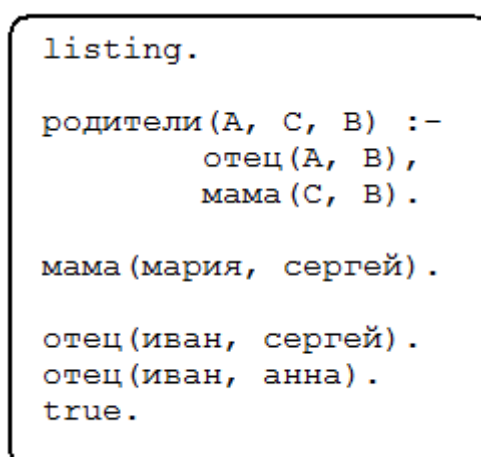



Рисунок 3.9 — База данных

Для закрытия сессии интерпретатора следует в окне консоли нажать кнопку  — **Завершить**.

Шаг 5. Выполнение запросов

Чтобы проверить работу программы вводите последовательно в окне консоли запросы (целевые утверждения), указанные в п. 3.3. При этом символы “?” вводить не требуется. Например, введем запрос

```
отец(иван, Р) .    % Кто дети Ивана?
```

SWI-Prolog выполнит интерпретацию этого целевого утверждения и в результате сопоставления с предикатами загруженной базы данных выведет первое значение переменной **Р** :

```
Р = сергей
```

Для получения следующего значения переменной **Р** введите символ “;” и нажмите **Enter**. SWI-Prolog выведет второе значение

```
Р = анна .
```

Чтобы проверить работу правила программы введите запрос (кто родители Сергея?):

```
родители(О, М, сергей) .
```

В результате доказательства этого целевого утверждения будут выведены следующие значения переменных **О** и **М**, при которых оно справедливо:


```
О = иван ,
М = мария
```

Если попытаться найти следующие значения переменных **О** и **М**, введя символ “;” и нажав **Enter**, то будет выведен ответ **false**. И это правильно, так как в нашей базе данных нет каких-либо других сведений о родителях Сергея.

Целевые утверждения можно вводить не только во время сеанса интерпретации, но и включать в исходный код программы во время её создания. В этом случае целевое утверждение должно начинаться с символов “?-“ (или “:-“). Добавьте в конец программы следующие строки:

```
% Целевое утверждение (запрос)
?-родители(О, М, сергей) , nl , write(О) , nl , write(М) .
```

Здесь **write** —это встроенный предикат вывода, а **nl** — встроенный предикат, обеспечивающий перевод курсора на новую строку.

Завершите предыдущий сеанс работы SWI-Prolog, нажав кнопку  — **Завершить**. Сохраните файл и выполните заново компиляцию программы. Теперь SWI-Prolog начнет сразу доказательство целевого утверждения, содержащегося в

программе. При этом условия (предикаты) целевого утверждения будут выполняться в порядке их записи.

Сначала SWI-Prolog выполнит поиск возможных вариантов сопоставлений для предиката **родители (О, М, сергей)**. При этом переменные **О** и **М** будут конкретизированы и получают значения: **иван** и **мария**. Предикаты **write** обеспечат вывод этих значений в окно консоли.

В примере программы использовались имена предикатов и констант, набранные литерами русского алфавита. SWI-Prolog поддерживает обработку таких имен. Однако в ряде случаев редактор среды Eclipse ProDT такие имена обрабатывает некорректно. Кроме этого, ухудшается переносимость программ. Поэтому при разработке программ, которые требуют дальнейшего сопровождения, лучше использовать литеры английского алфавита.

3.5. Интерпретатор SWI-Prolog

3.5.1. Помощь

В процессе работы со SWI-интерпретатором доступна “on-line”помощь. В среде Eclipse ProDT для этого можно использовать следующие встроенные предикаты:

- **apropos(Образец)** — отображает все предикаты, функции или разделы помощи, которые содержат в своем имени подстроку **Образец**;
- **explain(Объект)** — выводит текст с объяснением любого объекта данных Пролога.

Например, если в окне консоли ввести запрос **apropos(assert)**, то будут выведены все предикаты, которые содержат в своем описании подстроку **assert** (рис.3.10).

```
apropos(assert) .
assert/1           Add a clause to the database
asserta/1          Add a clause to the database (first)
assertz/1          Add a clause to the database (last)
assert/2           Add a clause to the database, give
reference
asserta/2          Add a clause to the database (first)
assertz/2          Add a clause to the database (last)
assertion/1        Make assertions about your program
true.
```

Рисунок 3.10 — Результаты выполнения запроса **apropos(assert)**

Если ввести в окне консоли **explain(assert)**, то будет выведен текст с объяснением встроенного предиката **assert** (рис. 3.11).

```

explain(assert).
"assert" is an atom
system:assert/2 is a built-in meta predicate
    Summary: ``Add a clause to the database, give reference''
system:assert/1 is a built-in meta predicate
    Summary: ``Add a clause to the database''
    Referenced from 1-th clause of online_help:line_index/2
true.

```

Рисунок 3.11 — Результаты выполнения запроса **explain(assert)**

3.5.2. История команд

SWI-интерпретатор хранит в специальном списке перечень запросов (команд), которые были введены ранее. Такой список называют историей команд. Таблица 3.1 содержит перечень команд, которые позволяют выбирать и повторно выполнять ранее введенные запросы. Также перемещаться по списку истории команд можно с помощью клавиш со стрелками вверх/вниз.

Таблица 3.1 — Доступ к истории команд

Команда	Назначение команды
!!.	Повторить последний запрос
! <i>номер</i> .	Повторить запрос с заданным номером
! <i>подстрока</i> .	Повторить последний запрос, начинающийся заданной подстрокой
! <i>?подстрока</i> .	Повторить последний запрос, содержащий заданную подстроку
^ <i>старая</i> ^ <i>новая</i> .	Заменить <старая> на <новая> в последнем запросе
! <i>номер</i> ^ <i>старая</i> ^ <i>новая</i> .	Заменить в запросе с номером <номер>
! <i>подстрока</i> ^ <i>старая</i> ^ <i>новая</i> .	Заменить в запросе, начинающимся заданной подстрокой
! <i>?подстрока</i> ^ <i>старая</i> ^ <i>новая</i> .	Заменить в запросе, содержащем заданную подстроку
h.	Показать историю команд
!h.	Показать эту таблицу

Например, если в окне консоли ввести команду “**h.**”, то получим список ранее введенных запросов (рис. 3.11).

```

h.
1  consult('D:/eclipse_prolog/relative/src/parents.pl').
2  отец(иван, Р) .
3  родители(О, М, сергей) .
4  apropos(assert) .
5  explain(assert) .

```

Рисунок 3.12 — История команд

Обратите внимание на первый запрос, который был автоматически сформирован при запуске SWI-интерпретатора. Этот запрос содержит встроенный предикат **consult**, который обеспечивает компиляцию и загрузку базы данных из файла.

3.5.3. Отладка программы

Двумя основными средствами проверки работы программы в Прологе являются *трассировка и контрольные точки*.

Для включения режима трассировки используется предикат **trace**. Для её отключения — предикат **notrace**. Трассировка позволяет пользователю наблюдать за ходом выполнения программы во время её работы. Предикат **trace** вводится непосредственно перед проверяемым участком программы или запросом, а предикат **notrace** — сразу после него.

SWI-Prolog позволяет пользователю выбирать режим трассировки, при котором информация о ходе трассировки выдается в четырех стандартных точках (портах) обработки утверждения:

Call (вызов) — точка вызова доказываемого утверждения;

Exit (выход) — точка успешного выхода при доказательстве утверждения;

Redo (передоказать) — точка повторного доказательства утверждения;

Fail (неудача) — точка с состоянием неудачи при доказательстве утверждения.

В дополнение к этим точкам SWI-Prolog предлагает точку (порт) **unify**, которая позволяет пользователю инспектировать результаты после сопоставления (унификации) головы утверждения. Также имеется порт **exception**, который показывает исключительные ситуации, вызванные предикатом **throw** [6].

Предикат **trace/0** (цифра после знака “/” обозначает количество аргументов предиката), введенный в окне консоли, обеспечивает трассировку следующего за ним запроса (целевого утверждения). В ходе трассировки отображается имя порта, текущая глубина рекурсивного вызова и очередная цель.

Рассмотрим пример трассировки. Для этого добавим в текст программы, изображенной на рис.3.5, предикат **дети(Отец)**, обеспечивающий вывод на экран имен всех детей для заданного отца [1]:

```
дети(Отец) :-
    отец(Отец, Ребенок), nl, write(Ребенок), fail;
    true.
```

Сохраним файл и заново запустим SWI-интерпретатор. Введем последовательно два запроса: **trace** и **дети(иван)**. Теперь при доказательстве утверждения **дети(иван)** при каждом нажатии клавиши **Enter** в окне консоли будут построчно выводиться сообщения трассировщика (рис3.12).

```

|:-trace.
true.
:-дети(иван).
  Call: (6) дети(иван) ? :-
    Call: (7) отец(иван, _G191) ? Exit: (7) отец(иван, сергей) ? |
    Call: (7) nl ?
  Exit: (7) nl ? |
    Call: (7) write(сергей) ? сергей
  Exit: (7) write(сергей) ? |
    Call: (7) fail ? Fail: (7) fail ? |
    Redo: (7) отец(иван, _G191) ? Exit: (7) отец(иван, анна) ? |
    Call: (7) nl ?
  Exit: (7) nl ? |
    Call: (7) write(анна) ? анна
  Exit: (7) write(анна) ? |
    Call: (7) fail ? Fail: (7) fail ? |
    Call: (7) true ? Exit: (7) true ? |
    Exit: (6) дети(иван) ?
true.
|notrace.
true.

```

Рисунок 3.13— Трассировка запроса

При этом в стандартных точках выдачи информации после знака “?” можно вводить команды трассировщика. Команды трассировщика обозначаются одной литерой. В таблице 3.2 приведены некоторые команды трассировщика SWI-Prolog [6].

Таблица 3.2 — Команды трассировщика SWI-Prolog

Команда	Название	Назначение команды
A	Alternatives	Показать все цели, имеющие альтернативы.
L	Listing	Отобразить список утверждений с помощью предиката <code>listing</code> .
a	abort	Вернуться на верхний уровень интерпретатора.
b	break	Рекурсивно запускает интерпретатор с верхнего уровня.
c	creep	Продолжить выполнение до следующего порта.
e	exit	Завершить работу интерпретатора.
f	fail	Установить результат “неудача” при доказательстве текущей цели.
g	goals	Показать список дочерних целей (стек выполнения)
h	help	Отобразить список допустимых команд трассировщика
i	ignore	Игнорировать текущую цель
l	leap	Продолжить выполнение до следующей контрольной точки.
n	no debug	Продолжить выполнение не в режиме отладки
r	retry	Возврат к call порту текущей цели
s	skip	Продолжить выполнение до следующего порта цели, пропустив все дочерние цели
u	up	Продолжить выполнение и сделать остановку при заходе в порт родительской цели (т.о. пропускается текущая цель и все вызовы дочерних целей).
+	Spy	Установить контрольную точку для текущего предиката
-	No Spy	Удалить контрольную точку для текущего предиката
/	find	Выполнить поиск порта. После ‘/’ следует указать имя порта (можно сокращенно). Например, /f — поиск порта fail; /c — поиск порта вызова; /a — поиск любого порта

В процессе отладки иногда возникает необходимость приостановить работу программы при доказательстве очередной цели (целевого утверждения). Для этого применяют контрольные точки. Остановка в контрольной точке позволяет проверить состояние параметров и, при необходимости, воздействовать на дальнейший процесс доказательства. При достижении контрольной точки запускается трассировщик, если система SWI-Prolog находится в режиме отладки (включить режим отладки можно с помощью предиката **debug**).

Для установки контрольных точек на предикатах, удовлетворяющих заданной спецификации применяется вызов **spy(спецификация предиката)**, для её удаления — **nospy(спецификация предиката)**. Например, если мы хотим установить контрольную точку на любом предикате **отец**, то следует применить вызов **spy(отец/2)** или **spy(отец)**. Пример установки и удаления контрольных точек при работе с интерпретатором SWI-Prolog изображен на рис. 3.13.

```

гему(уван).
сергеу
анна
true.

spy(отец/2).
% Spy point on отец/2
true.

гему(уван).
  Call: (7) отец(уван, _G173) ?
  Exit: (7) отец(уван, сергеу) ?      Call: (7) nl ?

  Exit: (7) nl ?      Call: (7) write(сергеу) ?
сергеу
  Exit: (7) write(сергеу) ?      Call: (7) fail ?
  Fail: (7) fail ?      Redo: (7) отец(уван, _G173) ?
  Exit: (7) отец(уван, анна) ?      Call: (7) nl ?

  Exit: (7) nl ?      Call: (7) write(анна) ?
анна
  Exit: (7) write(анна) ?      Call: (7) fail ?
  Fail: (7) fail ?      Call: (7) true ?
  Exit: (7) true ?      Exit: (6) гему(уван) ?
true.

debugging.
% Debug mode is on
% Spy points (see spy/1) on:
%      отец/2
% No traced predicates (see trace/1)
true.

nospy(отец/2).
% Spy point removed from отец/2
true.

debugging.
% Debug mode is on
% No spy points
% No traced predicates (see trace/1)
true.

```

Рисунок 3.14 — Установка и удаление контрольных точек

Предикат **debugging** выводит в текущий выходной поток сведения о режиме отладки, установленных контрольных точках и трассируемых предикатах. По умолчанию режим отладки отключен. Однако система переходит в режим отладки при первом вызове предикатов: **debug**, **trace** или **spy**. Для отключения режима отладки применяется предикат **nodebug**.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бондарев В.Н. Искусственный интеллект: учеб. пособие / В.Н. Бондарев, Ф.Г. Аде.— Севастополь : Изд-во СевНТУ, 2002. — 615с.
2. Введение в интегрированную среду разработки Eclipse CDT: методические указания к лабораторным работам по дисциплине “Основы программирования и алгоритмические языки” для студ. дневной и заочной формы обучения направления 09.03.02 — “Информационные системы и технологии”/ СевГУ; сост. В. Н. Бондарев, Т.И. Сметанина. — Севастополь: Изд-во СевГУ, 2015. — 40 с.
3. Введение в среду разработки Лисп-приложений Eclipse Cuspr: методические указания к лабораторным работам по дисциплине “Методы и системы искусственного интеллекта” для студ. дневной и заочной формы обучения направления 09.03.02 — “Информационные системы и технологии” / СевГУ ; сост. В. Н. Бондарев. — Севастополь : Изд-во СевГУ, 2015. — 44 с.
4. Казарин С.А. Среда разработки Java-приложений Eclipse (ПО для объектно-ориентированного программирования и разработки приложений на языке Java): учеб. пособие [Электронный ресурс] / С.А.Казарин, А.П.Клишин. — М.: Федеральное агентство по образованию, 2008. — 77 с. — Режим доступа: http://window.edu.ru/window_catalog/files/r58397/Eclipse_Java.pdf. — Последний доступ: 19.12.2011. — Название с экрана.
5. Rahimberdiev Askar Проект Eclipse [Электронный ресурс] / Askar Rahimberdiev // RSDN Magazine.— №4.— 2004. — Режим доступа: <http://www.rsdn.ru/?article/devtools/eclipse.xml>. — Последний доступ: 19.12.2011. — Название с экрана.
6. Wielemaker Jan SWI-Prolog 6.2.5 : Reference Manual [Электронный ресурс] / Jan Wielemaker .— Amsterdam: University of Amsterdam, 2012.— 442p. — Режим доступа: <http://www.swi-prolog.org/download/stable/doc/SWI-Prolog-6.2.5.pdf> — Последний доступ: 27.12.2012. — Название с экрана.

Приложение А (справочное) Пример проекта “Поиск в ширину”

Рассмотрим пример проекта **breath**, реализующего поиск пути на графе состояний методом в ширину. Полное описание программы приведено в [1]. Ниже приведено содержание файла **breath.pl**

```
/*
-----
ПРОГРАММА ПОИСКА ПУТИ НА ГРАФЕ СОСТОЯНИЙ МЕТОДОМ В ШИРИНУ
-----
Для поиска используется предикат
    в_ширину(ОТК,ЗКР,Решение), где
ОТК - список открытых вершин;
ЗКР - список закрытых вершин;
Решение - список обследованных вершин.

Указанные списки состоят из подсписков, в каждом из которых два элемента.
Первый элемент - это имя некоторой вершины графа состояний,
второй - имя ее родительской вершины.

Список Решение формируется таким образом, что его первый подсписок
в качестве первого элемента содержит целевую вершину.
Просмотрев список Решение от целевой вершины до стартовой вершины,
программа восстановит решающий путь.Для этого используется предикат
    решающий_путь(Решение, Цель, Старт, Теmpr, Реш_путь), где
Цель - целевая вершина;
Старт - начальная вершина;
Теmpr - вспомогательная переменная (для накопления результатов поиска);
Реш_путь - список, фиксирующий найденный путь от стартовой до целевой вершины.

Предикат "поиск", приведенный в конце программы
демонстрирует работу указанных выше предикатов
*/

% Описание графа состояний
/* -----
предикат после(Вершина1,Вершина2) указывает,
что после вершины1 следует дочерняя вершина2
*/
после(a,b).
после(a,c).
после(b,d).
после(b,e).
после(d,h).
после(e,i).
после(e,g1).
после(c,e).
после(c,g2).
после(c,z).

% Целевая вершина
цель(g1).

/*-----
Предикат поиска
в_ширину(ОТК,ЗКР,Решение), где
ОТК - список открытых вершин;
ЗКР - список закрытых вершин;
Решение - список обследованных вершин
```



```

*/
в_ширину ([ [O1|O2] |Ост_ОТК],ЗКР,Решение):-
    цель (O1),!,Решение=[ [O1|O2] |ЗКР].
в_ширину ([ [O1|O2] |Ост_ОТК],ЗКР,Решение):-
    bagof ([Z|O1], (после (O1,Z),
        not (элемент1 (Z,ЗКР)), not (элемент1 (Z,Ост_ОТК))), L),
    слияние (Ост_ОТК,L,Нов_ОТК),!,
    в_ширину (Нов_ОТК, [ [O1|O2] |ЗКР],Решение);
    в_ширину (Ост_ОТК,ЗКР,Решение).

/*-----
Предикат элемент1 (Вершина,Список)
позволяет установить, входит ли Вершина
в один из подсписков, образующих Список
*/
элемент1 (H, [ [H|T] |_]).
элемент1 (T, [ [H|T] |_]).
элемент1 (H, [ [Y|Z] |Ост]) :-элемент1 (H,Ост).

/*-----
Предикат слияние (Список1,Список2,Список3)
формирует Список3 слиянием первых 2-х списков
*/
слияние ([],L,L).
слияние ([H|T],W,[H|Z]) :-слияние (T,W,Z).

/*-----
Предикат найти (Элемент,Список1,Список2) выполняет поиск
заданного Элемента в Списке1, состоящем из подсписков.
Результатом поиска является Список2, представляющий хвостовую часть Списка1,
в первый подсписок которой входит искомый элемент
*/
найти (H, [ [H|T] |Ост], [ [H|T] |Ост]).
найти (H, [ [Y|T] |Ост],X) :-
    найти (H,Ост,X).

/*-----
Предикат просматривающий список Решение и возвращающий путь от
стартовой вершины до целевой:
    решающий_путь (Решение, Цель, Старт, Темп, Реш_путь), где
Цель - целевая вершина;
Старт - начальная вершина;
Темп - вспомогательная переменная, при вызове пустой список;
Реш_путь - список, фиксирующий найденный путь от стартовой до целевой вершины.
*/
решающий_путь (_,X,X,Темп,Реш_путь) :-Реш_путь=Темп.
решающий_путь (Решение,Цель,Старт,Темп,Реш_путь) :-
    найти (Цель,Решение, [ [H1|T1] |Ост1]),
    слияние (Темп, [ [H1|T1] ],Темп1),
    найти (T1,Ост1, [ [H2|T2] |Ост2]),
    слияние (Темп1, [ [H2|T2] ],Темп2),
    решающий_путь (Ост2,T2,Старт,Темп2,Реш_путь).

/*-----
Тестовый запрос
Найти решающий путь из начальной вершины "a" в конечную вершину "g1".
В результате обработки вопроса получим значения переменных:
Решение = [[g1 | e], [e | b], [d | b], [c | a], [b | a], [a | a]]
Реш_путь = [[g1 | e], [e | b], [b | a], [a | a]]
*/
поиск:-в_ширину ([ [a|a] ], [],Решение),nl,write ('Решение='),write (Решение),nl,
решающий_путь (Решение,g1,a, [],Реш_путь),write ('Реш_путь='),write (Реш_путь).

```

Приложение Б (справочное) Примеры операций ввода-вывода

```
% ПРИМЕРЫ ОПЕРАЦИЙ ВВОДА-ВЫВОДА
% Создайте новый проект "vvod_vyvod_files"
%-----

% Вывод списка на экран
/*
Предикат вывод_списка(Список)
обеспечивает вывод каждого элемента Списка в новой строке
*/
вывод_списка([]).
вывод_списка([H|L]):-write(H),nl,вывод_списка(L).

% Ввод с клавиатуры символьных констант
/* -----
В примере реализован поиск адресов в базе фактов
по вводимой фамилии. При вводе фамилии в конце необходимо ставить точку.
*/
адрес(петров,'Тенистая 10').
адрес(иванов,'Древняя 9/15').
адрес(сидоров,'Университетская 30').
поиск_адреса:-
    repeat,
    write('Поиск адреса (для завершения работы напечатайте: стоп.)'),nl,
    write('Введите фамилию (в конце поставьте точку):'),nl,
    read(X),
    (X='стоп',!;
     адрес(X,A),write('Адрес:'),write(A),nl,fail).

% Простейший калькулятор (ввод арифметических выражений)
/*-----
В примере реализован калькулятор, вычисляющий арифметические выражения
с помощью предиката is.
При вводе выражения в конце необходимо ставить точку.
*/
калькулятор:-
    write('Вычисление выражений (для завершения работы напечатайте: стоп.)'),nl,
    write('Введите арифметическое выражение (в конце поставьте точку):'),nl,
    read(X),
    (X='стоп',!;
     Y is X, write(Y),nl,калькулятор).

%Чтение и запись термов в файл
/*-----
В примере реализовано копирование термов из файла F1 в файл F2.
Термы в файле F1 должны завершаться точкой
При вызове предиката "обработка-файлов" указывайте полные пути доступа к файлам.
*/
обработка_файлов(F1,F2):-
    see(F1),           %назначение F1 стандартным входным потоком
    tell(F2),          %назначение F2 стандартным выходным потоком
    чтение_запись1,    %копирование термов до конца файла (вариант1)
    %чтение_запись2,   %копирование термов до конца файла (вариант2)
    seen,told.         %переключение стандартных потоков на терминал

%копирование термов до конца файла
%рекурсивный вариант
чтение_запись1:-
```

```

    read(T1),                                %чтение терма
    копирование(T1,T2),                      %копирование
    write_term(T2),                          %запись терма на новой строке
    T1\==end_of_file, чтение_запись1.        %рекурсивный вызов, если не конец файла
чтение_запись1:-!.                          %завершить чтение-запись

%копирование термов до конца файла
%вариант с использованием цикла с возвратом repeat
чтение_запись2:-
    repeat,                                %цикл, создающий точки возврата
    read(T1),                              %чтение терма
    копирование(T1,T2),                    %копирование
    write_term(T2),                        %запись терма на новой строке
    T1\==end_of_file,!.                   %проверка - если конец файла, стереть все точки возврата

копирование(end_of_file,end_of_file):-!.
копирование(T1,T2):-T2=T1,!.

write_term(end_of_file):-!.                %если конец файла в F2 ничего не записывается
write_term(Term):-write(Term),write(' '),nl,!.

% Тестовый вызов предиката обработка_файлов(F1,F2)
/*-----
Сначала создайте в папке проекта с помощью Eclipse файл "f1.txt".
Например, со следующим содержимым:
(5+3*2).
программирование_на_Прологе.
start(f1,f2).
Затем введите запрос с вызовом нижеследующего предиката (т.е. files.)
*/
files:-обработка_файлов('d:/Eclipse_prolog/vvod_vyvod_files/f1.txt',
                        'd:/Eclipse_prolog/vvod_vyvod_files/f2.txt').

% Просмотрите созданный файл "f2.txt", предварительно обновив содержимое папки
% проекта в среде Eclipse: Файл > Обновить. Содержимое "f2.txt" должно полностью
% совпадать с содержимым "f1.txt"

```

Заказ № _____ от «_____» _____ 2015г. Тираж _____ экз.
Изд-во СевГУ