ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ

НОВГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ ЯРОСЛАВА МУДРОГО

ФУНКЦИОНАЛЬНОЕ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Часть 2
Логическое программирование

Лабораторный практикум

ВЕЛИКИЙ НОВГОРОД
2007

УДК 681. 3. 06 (075. 8)
ББК 32. 973. 26 - 018. 2

Печатается по решению
РИС НовГУ

Рецензент
доктор технических наук, профессор **В. В. Геппенер**
(Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»)

Функциональное и логическое программирование. Ч. 2. Логическое программирование : Лабораторный практикум / Сост. Д. В. Михайлов, Г. М. Емельянов; НовГУ им. Ярослава Мудрого. — Великий Новгород, 2007. — 88 с.

Приводятся общие сведения и рекомендации по решению задач нечислового характера с применением концепции логического программирования на примере двух наиболее известных реализаций языка Пролог — Турбо-Пролог 2. 0 и Visual Prolog 5. 2. Обращается обобое внимание на решение задач моделирования интеллектуальных действий человека при обработке различных видов информации на ЭВМ. Содержатся типовые задания, позволяющие приобрести навыки написания и отладки программ на языке Пролог при построении интеллектуальных систем различного назначения.

Предназначено для студентов специальности 230105, а также других, в учебных планах которых предусмотрены аналогичные дисциплины.

УДК 681. 3. 06 (075. 8)
ББК 32. 973. 26 - 018. 2

СОДЕРЖАНИЕ

Введение

Светлой памяти Курашовой Екатерины Петровны, нашего бессменного научного консультанта, помощника и вдохновителя посвящается.

Современный этап развития вычислительной техники характеризуется расширением сфер ее применения. На первый план выдвигаются задачи интеллектуализации процессов машинной обработки информации, моделирования рассуждений человека-эксперта при решении прикладных задач из различных областей человеческого знания. Кроме того, актуальным является также полное и непротиворечивое представление самих знаний. Тенденция развития современных языков программирования показывает необходимость разработки предметно-ориентированных языков, максимально приближенных к Естественному Языку (ЕЯ). Интеллектуализация инструментального ПО немыслима без изучения различных аспектов языкового поведения человека, разработки и исследования математических моделей различных сторон языковой деятельности. В этих условиях инженер по направлению «Информатика и вычислительная техника» должен наряду с хорошей технической подготовкой уметь решать сложные научные задачи представления знаний о языке общения (ЕЯ) во взаимосвязи с предметными знаниями человека-эксперта, организации работы со знаниями, обучения интеллектуальных систем. Решение указанных задач немыслимо без привлечения декларативного подхода в программировании, позволяющего описывать решение задачи на языке соотношений между объектами в заданной предметной области.

Дисциплина «Функциональное и логическое программирование» для специальности 230105 относится к числу специальных дисциплин. Она включает в себя рассмотрение основных вопросов современной теории и практики логического и функционального программирования и опирается на учебные курсы : «Дискретная математика», «Математическая логика и теория алгоритмов», «Алгоритмические языки и программирование».

Лабораторный практикум по разделу «Логическое программирование» имеет две составные части.

Первая часть (работы $1-4$) включает :

— изучение основ логического программирования и методики описания задач различного характера средствами языка Пролог.
— изучение методов организации повторяющихся и рекурсивных вычислений в языке Пролог, а также детальное изучение приемов работы со списками.
— изучение работы механизма согласования целевого утверждения.

Вторая часть (работы $5-8$) предназначена для ознакомления с возможностями Пролога, актуальными для построения прикладных интеллектуальных систем.

Лабораторные работы ориентированы на две наиболее известные реализации языка Пролог : Turbo Prolog 2. 0 и Visual Prolog 5. 2.

ЛАБОРАТОРНАЯ РАБОТА №1

СТРУКТУРА ПРОГРАММЫ НА ПРОЛОГЕ

Цель работы.

Целью работы является изучение структуры программы на Прологе и интегрированной Пролог-среды (Турбо-Пролога либо Visual Prolog'a).

1.1 Краткие теоретические сведения.

1.1.1 Концепция языка Пролог.

Пролог является языком программирования, который обеспечивает решение задач, выраженных в терминах объектов и отношений между ними.

Отличительные особенности языка Пролог состоят в следующем :

— Для представления знаний используются фразы Хорна.
— Программа описывает логическую модель Предметной Области в виде фактов относительно свойств Предметной Области и отношений между этими свойствами + правила вывода новых свойств и отношений из уже заданных.
— В качестве единообразной структуры данных используется терм.
— Отсутствуют операторы присваивания, ветвлений, безусловных переходов, а также указатели, которые используются при определении динамических структур данных в традиционных процедурно-ориентированных языках.

Написание программы на языке Пролог включает следующие этапы:

1) Объявление некоторых фактов об объектах и отношениях между ними.
2) Определение некоторых правил, касающихся объектов и отношений между ними.
3) Формулировка вопросов об объектах и отношениях между ними.

Для объяснения смысла программы применяются три семантические модели : декларативная, процедурная модели и модель в виде абстрактной машины.

Процедурная трактовка Пролог-программы.

При процедурной трактовке Пролог-программы подчеркивается последовательность шагов, которые выполняет интерпретатор при обработке запроса. Здесь имеет значение порядок следования подцелей в правиле.

Множество фраз, имеющих одно и то же имя и одинаковое количество аргументов, рассматривается как процедура. При этом запрос (или подцель правила) является вызовом процедуры.

Достоинство : позволяет адекватно представлять фразы, в которых важен порядок следования подцелей. Пример — вывод сообщений на экран в определенной последовательности.

Недостаток : невозможность разъяснения смысла фраз, вызывающих побочные эффекты управления. Примеры : остановка выполнения запроса, удаление фразы из программы.

Декларативная трактовка Пролог-программы.

При декларативной трактовке Пролог-программы специфицируются истинностные значения конкретных случаев отношений.

Для декларативной модели фразы Пролога являются формулами логики предикатов 1-го порядка, записанными в форме фраз Хорна.

Достоинство : эффективность представления знаний ввиду близости к семантике логики предикатов.

Недостаток : невозможность адекватного представления фраз, в которых важен порядок следования подцелей.

Модель в виде абстрактной машины.

С позиций декларативной модели Пролог-программа есть описание логической структуры. При выполнении запроса интерпретатор применяет по отношению к множеству фраз Пролога некоторую стратегию решения задачи. С точки зрения вычислений эта стратегия может быть описана при помощи некоторой абстрактной машины.

В языке Пролог запрос и множество фраз программы имеет вычислительный смысл. Это проявляется в том, что они вызывают определенное поведение интерпретатора Пролога. Модель в виде абстрактной машины описывает смысл запроса и множества фраз через действия этой машины.

Действия такой абстрактной машины можно рассматривать как применение правила резолюции.

Достоинство : модель в виде абстрактной машины наиболее точная из трех рассматриваемых моделей.

Недостаток : большая зависимость от реализации языка.

1.1.2 **Факты и правила.**

Определение 1.1.1. *Факт понимается как запись того отношения, значение которого истинно.*

Форма записи факта :

имя_предиката(аргумент {, аргумент}).

При этом :

1) Все имена предикатов и аргументов должны начинаться со строчной латинской буквы.
2) Перечисление аргументов — через запятую.
3) Каждый факт должен заканчиваться точкой.
4) Количество аргументов и вид отношений (направления отношений) определяются программистом и не меняются при выполнении программы.

К примеру, факт :

who_likes_what(ivan, programming)

не эквивалентен факту :

who_likes_what(programming, ivan),

поскольку имеет место изменение направления отношения «Иван увлекается программированием» на «Программирование увлекается Иваном».

Сам тип отношений в Турбо-Прологе и Visual Prolog'е описывается в разделе predicates. К примеру, для рассматриваемого who_likes_what это будет who_likes_what(symbol, symbol).

При описании фактов и правил в разделе clauses программы на Турбо-Прологе (Visual Prolog'е) одноименные предикаты должны быть сгруппированы :

```
clauses
    who_likes_what(ivan, programming).
    who_likes_what(ivan, reading).
    who_likes_what(mary, reading).
```

В терминологии Пролога любая совокупность фактов (и правил) называется Базой Данных (БД).

Определение 1.1.2. *Под правилами в Прологе понимаются наиболее общие утверждения об объектах и отношениях между ними.*

Пролог-правило имеет вид фразовой формы :

заключение :— усл1, усл2, . . . , услN.

Данное выражение считается основным в Прологе. Языки, подобные Прологу, считаются языками типа «если-то» : заключение истинно, если истинными являются все условия, перечисленные в правой части.

Пример правила для рассмотренной выше БД :

```
common_interests(X, Y):—
        who_likes_what(X, Z),
        who_likes_what(Y, Z),
        X<>Y.
```

1.1.3 **Термы и операторы.**

Пролог-программа состоит из термов.

Определение 1.1.3. *Терм — это либо константа, либо переменная, либо структура (=составной объект). Терм записывается как последовательность литер.*

Определение 1.1.4. *Константы представляют собой имена конкретных (не абстрактных !) объектов, либо конкретных отношений. Существует два вида констант : атомарные символы (в том числе специальные символы) и целые числа.*

Определение 1.1.5. *Переменная служит для представления объекта, на который нельзя сослаться по имени.*

В Прологе переменная обозначает объект, значение которого может быть найдено. При этом имя переменной не связывается с какой-то конкретной областью памяти, как это делается в процедурных языках.

В отличие от атомов, имя переменной начинается с прописной буквы или знака подчеркивания. Если обозначаемый переменной объект в рассматриваемом контексте не является существенным, то переменная считается анонимной и обозначается символом «_».

Определение 1.1.6. *Переменная называется конкретизированной, если существует объект, который она обозначает. Если не известно, что именно обозначает переменная, то считается, что переменная не конкретизирована.*

Определение 1.1.7. *Под структурой (составным объектом) в Прологе понимается единый объект, который состоит из совокупности других объектов, называемых в этом случае компонентами :*

имя_функтора(компонента {, компонента}).

Иногда удобно записывать некоторые функторы как операторы.. В Прологе операторы не вызывают выполнения каких-либо арифметических операций.

Так, терм 3+4 — другой способ записи структуры +(3,4).

При построении арифметических выражений необходимо знание трех основных свойств каждого оператора :

— Позицию;
— Приоритет;
— Ассоциативность.

Операторы +,-,*,/ записываются между своими аргументами и называются инфиксными. Операторы «+» и «-» могут быть записанными перед своими аргументами. Пример : изменение знака : -x+y. В данной ситуации оператор «-» является префиксным.

Оператор, записываемый после своего аргумента, называется постфиксным. Пример : математическая запись факториала : $n!$

Определение 1.1.8. *Приоритет оператора определяет, какая операция выполняется первой.*

В Прологе каждый оператор связан со своим классом приоритета. Класс приоритета есть целое число, величина которого зависит от конкретной версии Пролога. Однако в любой версии оператор с большим приоритетом имеет класс приоритета, более близкий к 1.

От свойства ассоциативности операторов зависит порядок выполнения операторов с одинаковым приоритетом. Пример : Y=8/2/2. В подобных случаях необходимо знать, является ли данный оператор левоассоциативным или правоассоциативным.

Определение 1.1.9. *Левоассоциативный оператор должен иметь слева операции одинакового или более высокого приоритета, а справа — операции низшего приоритета.*

В Прологе арифметические операции (+,-,*,/) являются левоассоциативными. Это означает, что выражения, подобные Y=8/2/2 интерпретируются как Y=(8/2)/2.

1.1.4 **Вопросы.**

Запись вопросов в Прологе сходна с записью фактов и отличается местоположением. Для формулировки вопросов в программах на Турбо-Прологе и Visual Prolog'е существует раздел goal.

Обращение к Прологу с вопросом инициализирует процедуру поиска в базе данных, ранее введенной в систему. Пролог просматривает БД в поисках предиката, сопоставимого с вопросом.

Определение 1.1.10. *Предикаты считаются сопоставимыми, если они совпадают посимвольно и их соответствующие аргументы попарно совпадают.*

Если предикат вопроса сопоставим с предикатом одного из фактов/правил в БД, то считается, что вопрос согласуется с БД. При этом ответом на вопрос будет либо Yes, либо No.

Переменные в вопросах используются для того, чтобы найти какой-либо объект. Пример : для вопроса who_likes_what(ivan, X) ответом будет :

X=programming
X=reading 2 Solutions
Yes

В сложных вопросах, содержащих конъюнкцию, и в правилах Пролог при согласовании каждому целевому утверждению приписывает его собственный маркер. Если в базе данных есть факт, соответствующий целевому высказыванию, то Пролог отмечает найденное место и пытается согласовать оставшиеся целевые высказывания. Для каждой подцели просмотр БД начинается с начала.

1.1.5 **Согласование целевых утверждений.**

Для доказательства целевых утверждений Пролог использует известные утверждения. Если подцель представляет собой факт, то согласование заканчивается, как только факт будет найден в базе. Если подцель есть правило, то задача сводится к конъюнкции предикатов-подцелей. При согласовании Пролог руководствуется следующими правилами :

— Константа равна только самой себе.
— Переменная может быть конкретизирована любым объектом Пролога.
— Структуры совпадают, если совпадают их функторы, положение и количество аргументов.

Для каждой подцели Пролог генерирует свой маркер. Если целевое утверждение не доказано, возбуждается процесс возврата.

Как в теле правила, так и в вопросе Пролог пытается согласовать входящие в конъюнкцию целевые утверждения в том порядке, в каком они написаны (слева направо). Доказательство согласованности целевого утверждения с базой данных включает в себя поиск соответствующих (сопоставимых) утверждений, пометку этого места базы данных и затем доказательство возникших подцелей. В ходе доказательства согласованности целевых утверждений с базой данных происходит конкретизация переменных. Показанный процесс получил название поиска с возвратом.

Когда целевое утверждение недоказуемо, осуществляется возврат по «цепочке доказательств» в место выбора утверждения для согласования заново соответствующих целевых утверждений. При этом Пролог пытается найти альтернативное утверждение, соответствующее данной цели. Вначале происходит расконкретизация всех переменных, конкретизированных в ходе доказательства данного целевого утверждения. Затем возобновляется поиск в базе данных, начиная с помеченного маркером места. Если будет найдено другое утверждение, соответствующее целевому, Пролог помещает в это место маркер, поиск в базе данных продолжается, начиная с помеченного места.

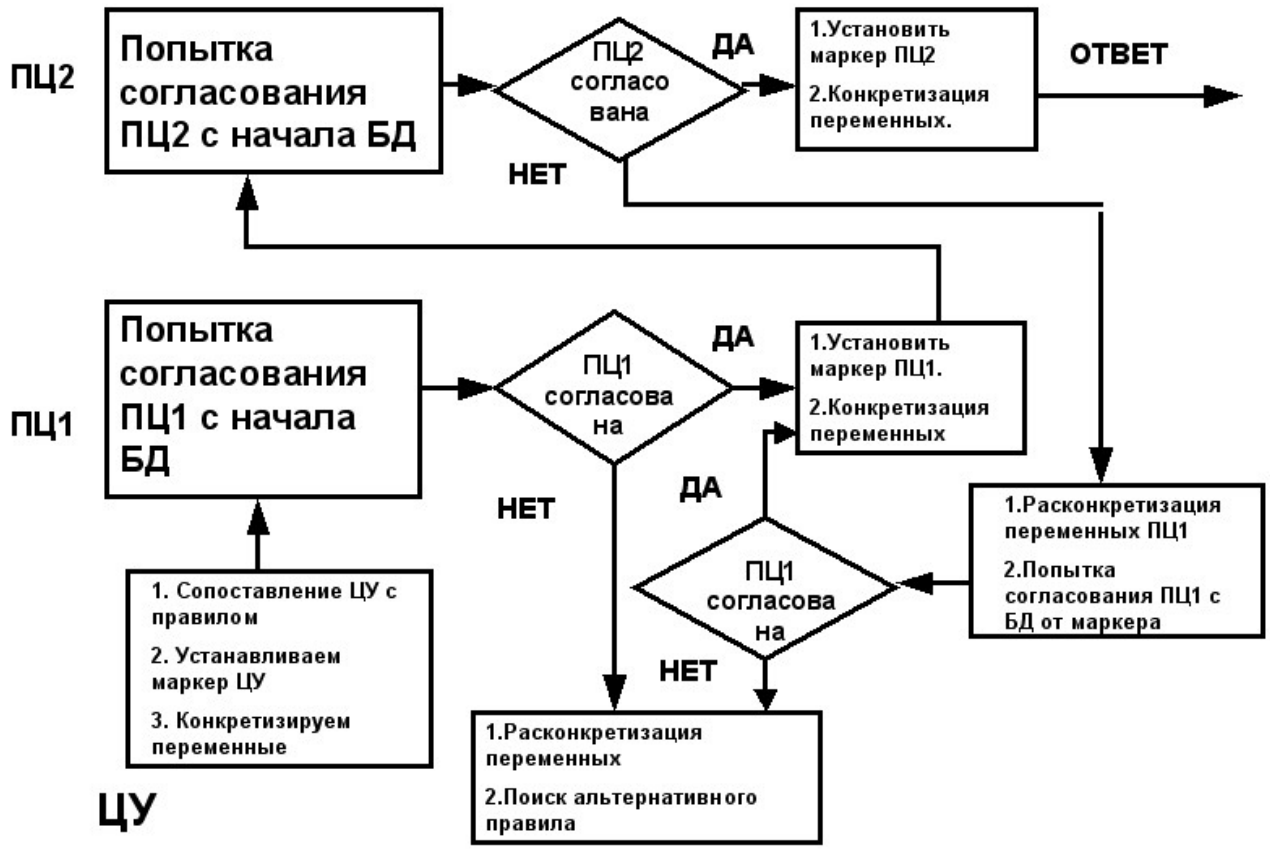

Рис. 1.1. Диаграмма согласования ЦУ, общий случай

Согласование целевых утверждений при наличии правил.

В качестве примера рассмотрим последовательность действий по согласованию целевого утверждения

common_interests(ivan, mary)

при наличии правила common_interests в рассмотренной выше БД. Факт common_interests(ivan, mary) в БД отсутствует, но есть соответствующее правило. Пролог отмечает это место в БД. При этом переменная X конкретизируется значением ivan, Y — значением mary.

Вначале Пролог ищет соответствие для предиката

who_likes_what(ivan, Z).

Факт, с которым происходит сопоставление, есть who_likes_what(ivan, programming), и тем самым первая цель достигнута. Пролог отмечает маркером соответствующее место в БД и записывает, что Z присвоено значение *programming*.

Затем Пролог пытается найти соответствие для следующего предиката в правиле, для чего ищет в базе данных факт who_likes_what(mary, programming) и, не находя его, расконкретизирует переменную Z. Теперь Пролог продолжает поиск соответствия для для предиката who_likes_what(ivan, Z) начиная от того места в БД, где находится маркер, и находит факт who_likes_what(ivan, reading). При этом Z присваивается значение *reading*. Утверждение who_likes_what (mary, reading) успешно согласуется с БД. Последняя цель в конъюнкции также успешно достигается и тем самым доказывается, что факт common_interests(ivan, mary) является истинным, Пролог отвечает Yes.

1.1.6 **Структура программы на Турбо-Прологе.**

Программа на Турбо-Прологе имеет следующие разделы :

— *domains* — для описания типов данных;
— *predicates* — для описания предикатов, которые используются в программе;
— *database* — для описания предикатов динамической базы данных;
— *clauses* — для описания фактов и правил (предикатов), составляющих программу;
— *goal* — для целевых утверждений (вопросов).

Разделы *predicates* и *clauses* должны присутствовать в любой программе, раздел *domains* используется для описания пользовательских типов и списков. Если в программе используются только стандартные типы данных : *symbol*, *string*, *char*, *integer*, *real*, *file*, то этот раздел может быть опущен.

Раздел *database* должен присутствовать только в тех программах, которые работают с динамической базой данных, т. е. изменяют программу в процессе ее работы, в противном случае этот раздел опускается.

Раздел *goal* должен содержать сформулированное на Прологе назначение программы и часто этот раздел также опускается. Комментарии могут помещаться в любом месте программы, на их длину нет ограничений, обрамляются комментарии символами /* ... */.

1.1.7 **Основные разделы Visual Prolog-программ.**

Синтаксис Visual Prolog-программ в общих чертах сходен с синтаксисом Турбо-Пролога.

Обычно программа на Visual Prolog'е состоит из четырех основных программных разделов :

— раздела *clauses* (предложений);
— раздела *predicates* (предикатов);
— раздела *domains* (доменов);
— раздела *goal* (целей).

Помимо основных разделов, в программах на Visual Prolog'е часто используются разделы фактов *(facts)*, констант *(constants)* и различные глобальные *(global)* разделы.

Раздел фактов.

В ряде случаев в процессе работы программы необходимо модифицировать (изменить, удалить или добавить) некоторые из фактов, с которыми она работает. При этом факты рассматриваются как динамическая или внутренняя база данных, которая при выполнении программы может изменяться. Для объявления тех фактов программы, которые рассматриваются как части динамической (или изменяющейся) базы данных, в Visual Prolog-программу включают специальный раздел — *facts*. Ключевое слово *facts* в Visual Prolog'е является синонимом ключевого слова *database* в Турбо-Прологе.

Раздел констант.

В Visual Prolog-программах используется следующий синтаксис объявления констант :

<Id>=<Макроопределение>,

где <Id> — имя символической константы, <Макроопределение> — присваиваемое ей значение.

Перед компиляцией программы Visual Prolog заменяет каждую константу на соответствующую ей строку.

Глобальные разделы.

Visual Prolog позволяет объявить некоторые разделы *domains*, *predicates*, *clauses* глобальными, для этого перед названием раздела помещается ключевое слово *global* : *global domains*, *global predicates*, *global facts*.

Директивы компилятора.

Visual Prolog поддерживает несколько директив компилятора, которые можно добавлять в программу для сообщения компилятору специальных инструкций по обработке программы во время компиляции.

Так, наиболее часто используемые в программе процедуры описываются в отдельном файле, а для вызова их из других файлов программы применяется директива include :

include «myfile.pro»

Большинство директив компилятора можно устанавливать с помощью меню среды визуальной разработки Visual Prolog : **Options | Project | Compiler Options** .

1.2 Задание на лабораторную работу.

1.2.1 Вариант 1 — для Турбо-Пролога.

Запустите Турбо-Пролог (файл prolog.exe). Изучите интегрированную среду Турбо-Пролога, обращая внимание на отличия от среды Турбо-Паскаля.

Выполните упражнения из раздела 1.2.3. Выясните назначение и возможности каждой программы.

1.2.2 Вариант 2 — для Visual Prolog'a.

Запустите Visual Prolog. При запуске следуйте указаниям на стр. 200 в [1].

Изучите визуальную среду разработки (VDE, Visual Development Environment). При этом обратить внимание на работу эксперта приложений, эксперта кода, браузера исходного кода, интерактивной справочной системы.

Создайте тестовый проект и выполните программу «Hello, world !», следуя указаниям на стр. 201–205 в [1] :

1) Создайте новый проект. Для этого выберите команду **Project | New Project**, активизируется диалоговое окно **Application Expert**.
2) Определите базовый каталог и имя проекта. Имя в поле **Project Name** следует определить как «TestGoal». Установите флажок Multiprogrammer Mode и щелкните мышью внутри поля **Name of . PRJ File**. При этом внутри данного поля появится имя файла проекта **TestGoal.prj** (рис.1.2).

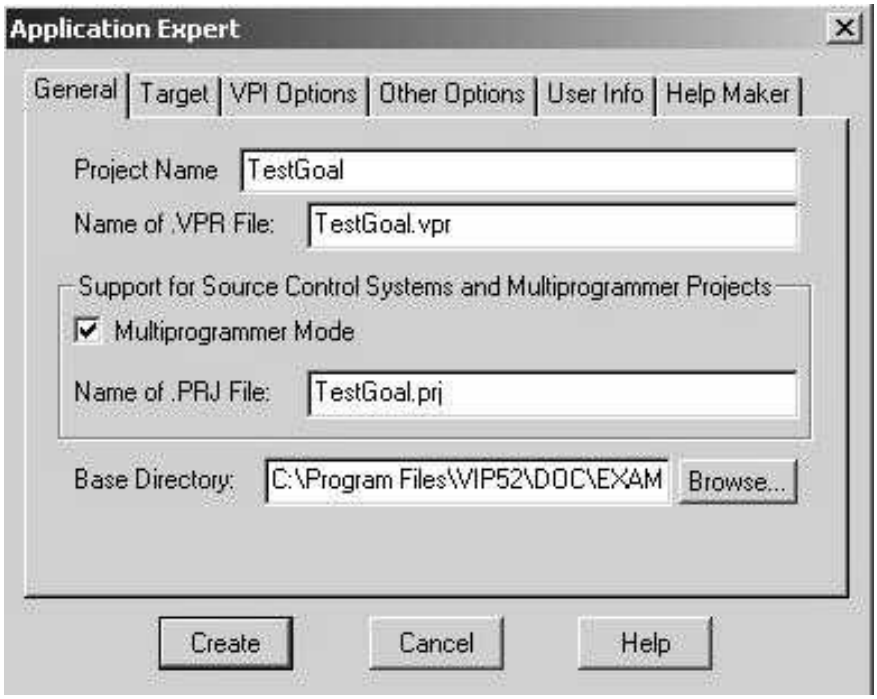

Рис. 1.2. Общие установки проекта в окне **Application Expert**

3) Определите цель проекта, для чего на вкладке **Target** для нашего тестового проекта следует выбрать параметры, отмеченные на рис.1.3. После этого нажмите кнопку **Create**, чтобы создать файлы проекта по умолчанию.

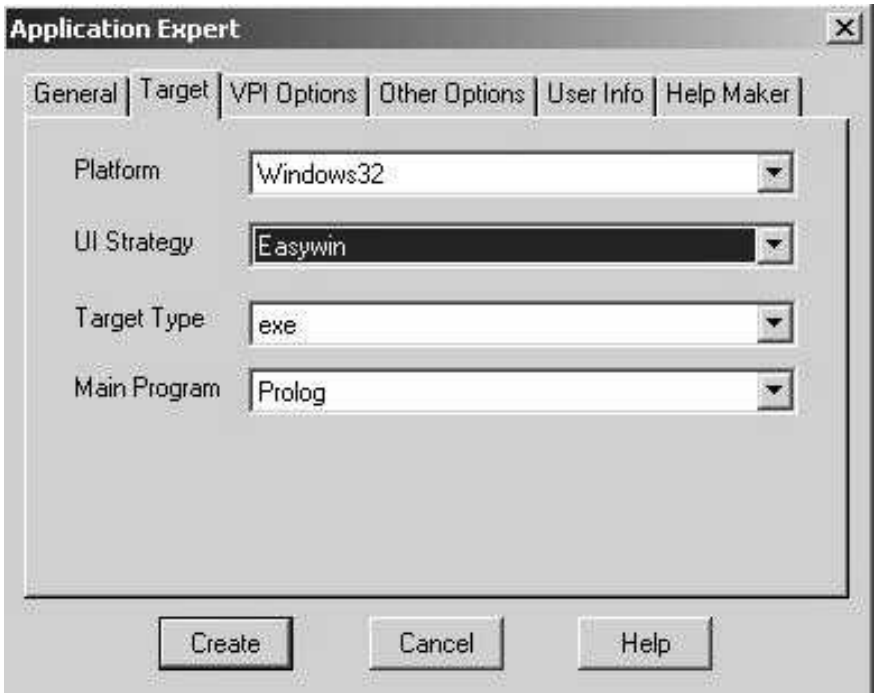

Рис. 1.3. Параметры цели проекта

4) Установите требуемые опции компилятора для созданного TestGoal-проекта, для чего в главном меню среды визуальной разработки следует выбрать команду **Options | Project | Compiler Options**. При этом активизируется окно **Compiler Options**. В этом окне откройте вкладку **Warnings** (рис.1.4).

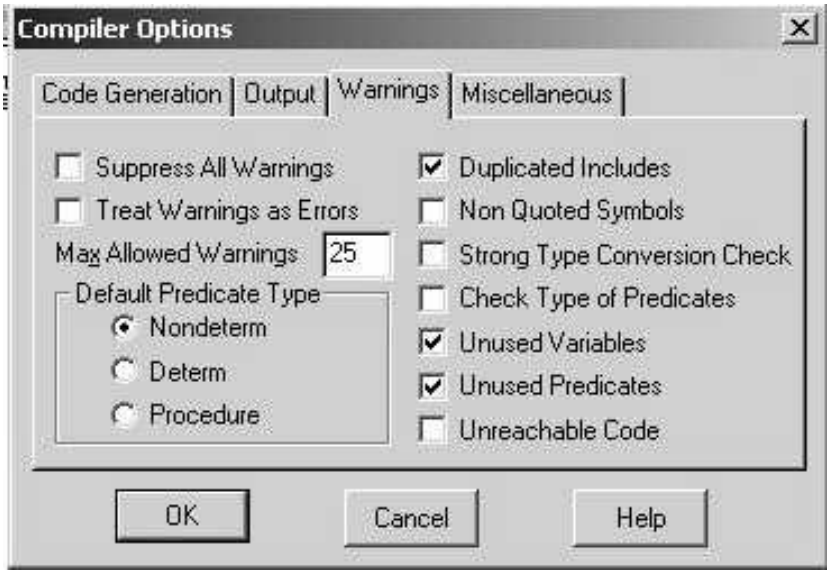

Рис. 1.4. Установки опций компилятора

5) Выполните следующие действия :

— Установите переключатель **Nondeterm**. При этом компилятором Visual Prolog'a по умолчанию принимается, что все определяемые в программе предикаты являются недетерминированными, то есть могут породить более одного решения.
— Снимите флажки **Non Quoted Symbols**, **Strong Type Conversion Check** и **Check Type of Predicates**. Это будет подавлять некоторые возможные предупреждения компилятора, которые не важны для понимания выполняемого примера.
— Нажмите кнопку OK, чтобы сохранить установки опций компилятора.

6) Создайте новый файл программы, для чего в главном меню среды визуальной разработки выберите команду **File | New**. При этом на экране появляется новое окно редактирования с именем **noname.pro** (рис.1.5).

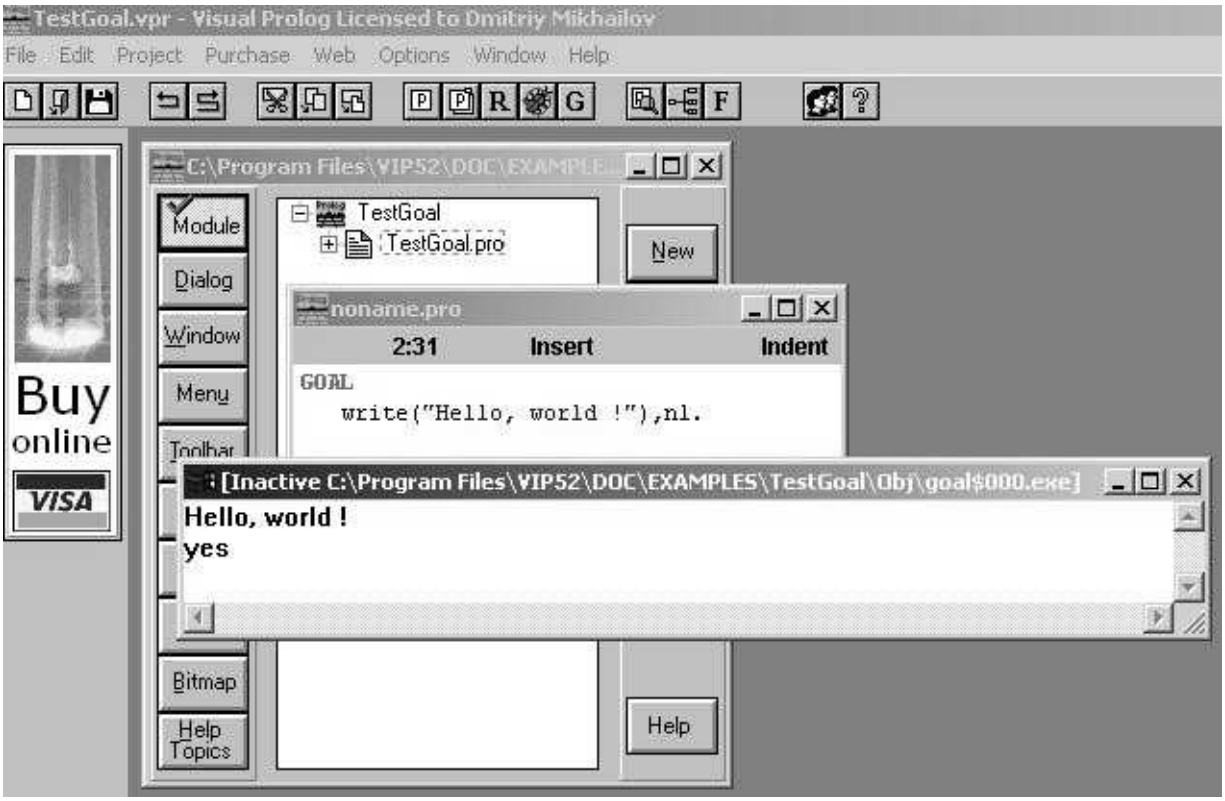

Рис. 1.5. Тестовая программа «Hello, word !»

7) В появившемся окне напечатайте следующий программный код :

Goal write(«Hello, world !»),nl.

Для выполнения введенной программы необходимо в главном меню среды визуальной разработки выбрать команду **Project | Test Goal**. Результат ее работы представлен на рис.1.5.

После создания и апробации тестового примера «Hello, word!» выполните **Упражнения 2–6** из раздела 1.2.3, следуя указаниям на стр. 205 в [1]. При этом, находясь в проекте TestGoal, следует использовать команду **File | Open** главного меню для открытия файлов упражнений и команду **Project | Test Goal** (или комбинацию клавиш <Ctrl>+<G>) для тестирования загружаемых примеров.

1.2.3 **Упражнения.**

Упражнение 1.

Запустите приведенную ниже программу и выясните, как она выполняется :

```
predicates
        male(symbol).
        parent(symbol, symbol, symbol).
        son(symbol, symbol).
clauses
        male(ivan).
        male(tom).
        male(john).
        parent(tom, ivan, mary).
        parent(john, ivan, mary).
        parent(anna, ivan, mary).
        parent(olga, ivan, mary).
        son(X, Y):- male(X), parent(X, Y, _).   /* X — сын, Y — отец*/
        son(X, Y):- male(X), parent(X, _, Y).   /* X — сын, Y — мать*/
```

Упражнение 2.

Дана программа, не содержащая разделов *domains* и *goal* :

```
predicates
        common_interests(symbol, symbol)
        who_likes_what(symbol, symbol)
        likes(symbol, symbol)
clauses
        common_interests(X, Y):-
                who_likes_what(X, Z), Z<>X,
                who_likes_what(Y, Z), X<>Y.
        who_likes_what(ivan, reading).
        who_likes_what(peter, programming).
        who_likes_what(peter, reading).
        who_likes_what(mike, tennis).
        who_likes_what(mike, programming).
        likes(X, Y):-
                who_likes_what(X, Y);
                common_interests(X, Y).
```

Запустите программу и выясните :

— что нравится каждому из упомянутых людей;
— кто и с кем имеет общие интересы;
— кто увлекается программированием.

Упражнение 3.

Дана программа :

```
domains
    title, author = symbol
    pages = integer
predicates
    book(title, pages)
    written_by(author, title)
    long_novel(title)
clauses
    written_by(fleming, «DR NO»).
    written_by(melville, «MOBY DICK»).
    book(«MOBY DICK», 250).
    book(«DR NO», 310).
    long_novel(Title):—
        written_by(_, Title),
        book(Title, Length),
        Length > 300.
```

Выясните назначение этой программы. Дополните программу информацией о книгах, изданных на русском языке. Добавьте в программу правила :

— отыскивающее самую тонкую книгу в БД;
— отыскивающее самую толстую книгу в БД;
— отыскивающее все книги в БД, не являющиеся самой тонкой и самой толстой;
— определяющее наличие самой тонкой и самой толстой книги в Базе Данных среди произведений заданного автора;
— определяющее самую тонкую и самую толстую книгу для заданного автора;
— отыскивающее в БД для заданного автора все книги, которые он не писал.

Упражнение 4.

«Новый магазин».

Имеется множество названий магазинов, среди которых есть существующие в Вашем городе. Имеется также множество товаров, среди которых имеются экзотические, которых нет на прилавках магазинов Вашего города. Предположим, что Вы разрабатываете проект нового магазина, торгующего экзотикой. Требуется :

— из множества названий магазинов выбрать для нового магазина то название, которым не назван ни один из существующих в Вашем городе;
— из множества всех возможных товаров выбрать экзотические, которые будут продаваться в новом магазине.

Написать программу, которая на основе фактов, описывающих магазины и товары (включая бинарное отношение «имеется в продаже в магазинах города»), находила бы название для нового магазина и определяла бы те товары, которые будут в нем продаваться.

Упражнение 5.

Решить задачу из Таблицы 1.1.

Таблица 1.1. Варианты заданий

Вариант	Задание
1	Четыре девочки : Надя, Вера, Галя и Люба, нарисовали по одному животному. Получились две лошади, одна овца и одна корова. Что нарисовала каждая девочка, если : — Надя и Вера не рисовали овцу; — Галя не рисовала корову; — Люба не рисовала лошадь; — Вера и Галя, Галя и Люба, Надя и Люба нарисовали разных животных.
2	«Грибники и ягодники». Три девочки : Оля, Лена и Маша, пошли в лес. Двое из них собирали грибы, одна — ягоды. Что собирала каждая девочка, если Оля с Леной и Маша с Леной не собирали одно и то же.
3	Четверо студентов : Иван, Петр, Михаил и Сергей, участвовали в олимпиадах по математике, физике и программированию. Определить вид дисциплины, в которой участвовал каждый студент, если : — Сергей и Петр не участвовали в олимпиаде по программированию; — Михаил не участвовал в олимпиаде по математике; — Иван не участвовал в олимпиаде по физике; — Сергей и Иван выступали в одной дисциплине; — Каждый студент участвовал ровно в одной дисциплине.
4	Четверо друзей : Иван, Петр, Михаил и Сергей, проводили свободное время по-разному : двое играли в шахматы, один читал книги, один смотрел телевизор. Определить, кто каким образом проводил время, если Сергей не играл в шахматы, а Петр не смотрел телевизор.
5	Четверо друзей : Иван, Петр, Михаил и Сергей, проводили летние каникулы по-разному : один был в деревне, один оставался дома, один был на море, один — в оздоровительном лагере. Определить, кто каким образом отдыхал, если :

Продолжение таблицы 1.1

Вариант	Задание
	— Иван, Петр и Михаил не были в оздоровительном лагере; — Иван и Михаил не были в деревне; — Михаил не оставался дома.
6	Иван, Петр, Михаил и Сергей посвящали свое свободное время чтению книг. Один читал о путешествиях, другой — о войне, третий — о спорте, четвертый — детектив. Кто о чем читал, если : — Иван и Петр не читали о войне; — Михаил и Сергей не читали о спорте; — Петр и Михаил не читали детектив; — Иван и Сергей не читали о путешествиях.
7	Четыре девочки : Надя, Вера, Галя и Люба, сажали плодовые деревья. Одна сажала яблони, другая груши, третья сливы, четвертая — вишни. Что сажала каждая девочка, если : — Вера не сажала яблони и груши; — Галя не сажала яблони и сливы; — Люба не сажала сливы и вишни; — Надя не сажала груши и вишни.

Упражнение 6.

Решить задачу из Таблицы 1.2.

Таблица 1.2. Варианты заданий

Вариант	Задание
1	Можно ли расставить на гранях куба целые числа от 1 до 6 так, чтобы каждое число являлось делителем суммы своих соседей ?
2	Расшифруйте числовой ребус : Т>Р>А>Н<С<П<О<Р<Т>И>Р>О<В<К<А (разные буквы обозначают разные цифры)
3	Расшифруйте числовой ребус : БЕЕЕ - М = МУУУ (одинаковым буквам соответствуют одинаковые цифры, разным — разные).
4	Расшифруйте числовой ребус : т р и - д в а = я р д (одинаковым буквам соответствуют одинаковые цифры, разным — разные).

Продолжение таблицы 1.2

Вариант	Задание
5	Расшифруйте числовой ребус : $\begin{cases} \text{МА*МА=МИР} \\ \text{АМ*АМ=РИМ} \end{cases}$ (одинаковым буквам соответствуют одинаковые цифры, разным — разные).

1.3 **Содержание отчета по работе.**

Отчеты по лабораторным работам в дальнейшем оформляются в соответствии с требованиями Единой Системы Программной Документации (ЕСПД), а также СТП НовГУ (в частности, СТП 1. 302-96 и СТП 1. 102).

Отчет по Лабораторной работе № 1 должен содержать :

1) Формулировку цели и задач работы.
2) При выполнении работы в среде Турбо-Пролога — сравнительный анализ интегрированных сред Турбо-Паскаля и Турбо-Пролога.
3) При выполнении работы в среде Visual Prolog — сравнительный анализ среды визуальной разработки Visual Prolog'a и визуальных сред других известных на сегодняшний день языков программирования (Visual C++, Borland Delphi и т. п.).
4) Протоколы работы с программами в соответствии с заданиями в из раздела 1.2.3.
5) Описание процесса разработки всех программ по заданию лабораторной работы.
6) Тексты программ для всех выполненных заданий с комментариями и обоснованиями.
7) Выводы с обоснованием полученных результатов и рекомендациями по дальнейшему совершенствованию разработанных Пролог-программ.

ЛАБОРАТОРНАЯ РАБОТА №2

ОРГАНИЗАЦИЯ ПОВТОРЯЮЩИХСЯ И РЕКУРСИВНЫХ ВЫЧИСЛЕНИЙ

Цель работы.

Целью работы является изучение методов организации повторяющихся и рекурсивных вычислений средствами языка Пролог.

2.1 Краткие теоретические сведения.

2.1.1 Рекурсия как основной метод программирования на Прологе.

Рекурсивное правило содержит само себя в качестве компоненты. В отличие от рекурсии нерекурсивные повторяющиеся нерекурсивные вычисления в Прологе организуются при помощи встроенных предикатов fail и !(отсечение).

В общем случае рекурсивное правило имеет следующий вид :

```
recursive_rule(<фактические параметры через запятую>):—
    <предикаты и правила>,
    recursive_rule(<фактические параметры рекурсивного вызова>).
```

Для передачи значений между правилами используется стек. Всякий раз при рекурсивном вызове новые копии используемых значений помещаются в стек. В Турбо-Прологе и Visual Prolog'е имеются средства для автоматического освобождения использованной части стека.

Любое рекурсивное правило должно включать в себя как минимум по одной из следующих компонент :

1) Условие окончания рекурсии в виде нерекурсивного правила (факта).
2) Изменяющийся аргумент рекурсивного вызова. При этом положение рекурсивного вызова в теле правила может быть любым.

Факт или факты, обеспечивающие завершение рекурсии, должны в программе помещаться перед правилом, а не после него во избежание левосторонней рекурсии.

Левосторонняя рекурсия возникает в случае, когда правило порождает подцель, эквивалентную исходной цели, которая явилась причиной использования этого правила. В процедуре с левой рекурсией рекурсивная подцель стоит слева от других подцелей.

Пример :

```
dog(X):—
        dog(Y),
        parent(Y, X).
dog(reks).
```

При попытке согласовать целевое утверждение dog(X) Пролог вначале пытается использовать правило и рекурсивно порождает подцель dog(Y). Попытка найти соответствие этой цели вновь приводит к выбору первого правила и так далее. Причина зацикливания заключается в отсутствии возможности использования механизма возврата. Для того, чтобы начался возврат, Пролог должен потерпеть неудачу при проверке первого утверждения, чего не происходит в приведенном примере.

2.1.2 **Вычисление факториала как пример рекурсии.**

Количество аргументов : 1.
Тип аргумента — целое число.
Условие выхода из рекурсии — факториал 0.

Вид рекурсивного правила :

```
fact(0, 1).
fact(N, M):—
        N1 = N - 1,
        fact(N1, M1),
        M = N * M1.
```

Рассмотрим работу нашего правила для вычисления 4!.

Вначале Пролог пытается выполнить подцель fact(4, M). Программа пытается сопоставить подцель с подправилом fact(0, 1). Сопоставление неудачно. Затем следует попытка сопоставления подцели с fact(N, M). На этот раз сопоставление завершается успешно с присвоением переменной N значения 4. В этом правиле переменной N1 присваивается значение 3, то есть значение N-1. Затем правило вызывает самого себя в виде fact(3, M). После этого вызова в теле правила идет вычисление значения переменной M с использованием свободной переменной M1, представляющей промежуточное значение факториала. Однако поскольку только что был вызван рекурсивный процесс, значение переменной M1 не может быть вычислено.

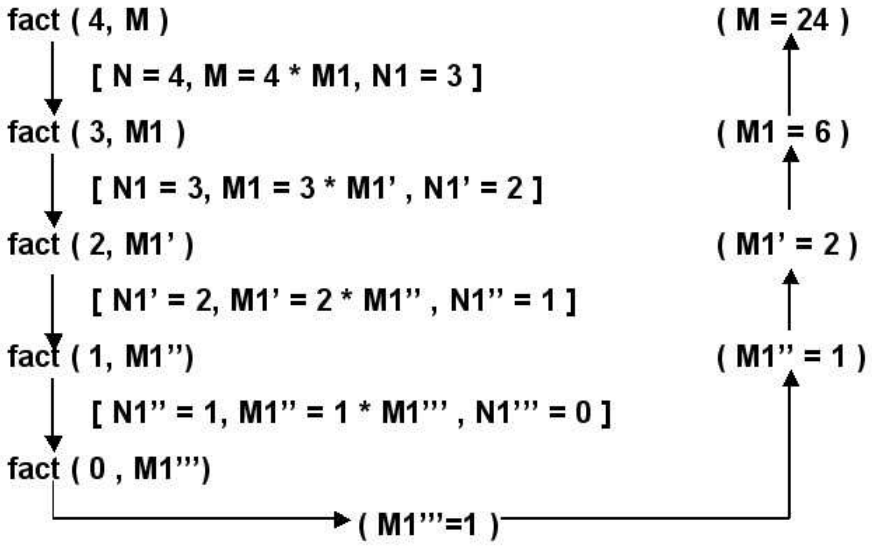

Рис. 2.1. Этапы решения задачи «Вычисление факториала»

Этот циклический процесс сопоставления продолжается до тех пор, пока не будет получено fact(0, M1). Теперь это правило сопоставляется с fact(0, 1) и M1 конкретизируется значением 1. При развертывании рекурсии программа запоминала значение M для последующего вычисления уже после достижения условия окончания (выхода из рекурсии), при извлечении рекурсивных вызовов из стека, рис.2.1.

2.1.3 **Числа Фибоначчи и треугольник Паскаля.**

Числа Фибоначчи определяются с помощью следующего рекуррентного соотношения :

$$fib_0 = 0, \quad fib_1 = 1 \quad \text{и} \quad fib_{i+1} = fib_i + fib_{i-1} \quad \text{для} \quad i > 0 \tag{2.1}$$

Последовательность чисел Фибоначчи начинается так :

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

Для всех неотрицательных чисел n и k функция C_n^k :

$$C_n^k = \begin{cases} \frac{n!}{k!(n-k)!} & \text{для} \quad 0 \le k \le n \\ 0 & \text{для} \quad 0 \le n < k \end{cases} \tag{2.2}$$

называется биномиальным коэффициентом. Читается : C из n по k (или n над k).

Значения биноминальных коэффициентов могут быть последовательно определены с помощью так называемого треугольника Паскаля :

$$\begin{array}{ccccccccc} & & & & C_0^0 & & & & \\ & & & C_1^0 & & C_1^1 & & & \\ & & C_2^0 & & C_2^1 & & C_2^2 & & \\ & C_3^0 & & C_3^1 & & C_3^2 & & C_3^3 & \\ C_4^0 & & C_4^1 & & C_4^2 & & C_4^3 & & C_4^4 \end{array}$$

Утверждение 1. *В треугольнике Паскаля каждый биноминальный коэффициент C_n^k равен сумме двух стоящих над ним коэффициентов : C_{n-1}^{k-1} (слева) и C_{n-1}^k (справа) :*

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k \tag{2.3}$$

Доказательство. В соответствии с определением 2.2 :

$$C_{n-1}^{k-1} = \frac{(n-1)!}{(k-1)! \times (n-1-k+1)!} = \frac{(n-1)!}{(k-1)! \times (n-k)!} \tag{2.4}$$

В то же время

$$C_{n-1}^k = \frac{(n-1)!}{k! \times (n-1-k)!} \tag{2.5}$$

Умножим числитель и знаменатель в (2.5) на $(n-k)$. Получаем :

$$C_{n-1}^k = \frac{(n-k) \times (n-1)!}{k! \times (n-k) \times (n-1-k)!} = \frac{(n-k) \times (n-1)!}{k! \times (n-k)!} \tag{2.6}$$

Умножаем числитель и знаменатель в (2.4) на k. Получаем :

$$C_{n-1}^{k-1} = \frac{k \times (n-1)!}{k \times (k-1)! \times (n-k)!} = \frac{k \times (n-1)!}{k! \times (n-k)!} \tag{2.7}$$

Складываем (2.6) и (2.7). Получаем :

$$C_{n-1}^{k} + C_{n-1}^{k-1} = \frac{(n-k) \times (n-1)! + k \times (n-1)!}{k! \times (n-k)!} =$$

$$= \frac{(n-1)! \times (n-k+k)}{k! \times (n-k)!} = \frac{n \times (n-1)!}{k! \times (n-k)!} = \frac{n!}{k! \times (n-k)!},$$

что соответствует определению 2.2 функции C_n^k. Таким образом, для $0 \leq k \leq n$ имеет место рекурсивное соотношение :

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k, \tag{2.8}$$

что и требовалось доказать.

2.2 Задание на лабораторную работу.

Задание 1.

Написать программы :
— генерации последовательности N первых членов ряда Фибоначчи;
— генерации таблицы чисел для треугольника Паскаля.

При генерации таблицы чисел для треугольника Паскаля рекомендуется использовать соотношение 2.8. Генерацию производить с 0-й по n-ю строки включительно. Матрица биноминальных коэффициентов должна выводиться построчно.

Задание 2.

Написать программу вычисления суммы n первых членов бесконечного ряда. В зависимости от номера варианта взять ряд из задания в сборнике [10]. Номер задания для каждого варианта указан в таблице 2.1.

Таблица 2.1. Варианты заданий

Вариант	Номер задания	Вариант	Номер задания
1	3026	21	3007
2	2987	22	3008
3	2988	23	3009
4	2990	24	3010
5	2991	25	3011
6	2992	26	3012
7	2993	27	3013
8	2994	28	3014

Продолжение таблицы 2.1

Вариант	Номер задания	Вариант	Номер задания
9	2995	29	3015
10	2996	30	3016
11	2997	31	3017
12	2998	32	3018
13	2999	33	3019
14	3000	34	3020
15	3001	35	3021
16	3002	36	3022
17	3003	37	3023
18	3004	38	3024
19	3005	39	3025
20	3006	40	2986

Задание 3.

«Консультант по транспорту».

Написать программу, которая должна находить все возможные способы перемещения по заданному городу на заданном виде общественного транспорта. Город и вид общественного транспорта задает преподаватель.

В качестве исходных данных выбираются начальные и конечные точки перемещения. Движение на всех видах общественного транспорта должно осуществляться в обе стороны. Программа должна находить решения следующим образом :

— если возможна поездка без пересадки, указывается номер маршрута заданного вида общественного транспорта (для метро — название линии (Кировско-Выборгская, Московско-Петроградская и т. п.));
— если без пересадки доехать нельзя, то проверяется возможность поездки с одной пересадкой;
— если нет возможности проезда ни без пересадок, ни с одной пересадкой, то проверяется возможность поездки с двумя пересадками, и т. д.

2.3 Содержание отчета по работе.

Отчет по работе должен содержать :

1) Формулировку цели и задач проводимых исследований.
2) Для каждого задания — анализ задания, обзор методов решения, выбор метода решения с обоснованием, описание процесса разработки программ, полученные результаты и их анализ (обоснование).
3) Тексты программ для всех выполненных заданий с комментариями и обоснованиями.
4) Для задания 3 — необходимые графические иллюстрации.
5) Выводы по проведенным машинным экспериментам.

ЛАБОРАТОРНАЯ РАБОТА №3

РАБОТА СО СПИСКАМИ

Цель работы.

Целью работы является изучение приемов работы со списками в Прологе, а также более детальное изучение рекурсивного программирования применительно к обработке списков.

3.1 Краткие теоретические сведения.

3.1.1 Список как частный вид структуры.

Определение 3.1.1. *Под списком понимается упорядоченная последовательность элементов, которая может иметь произвольную длину.*

Признак «упорядоченный» указывает на то, что порядок элементов в последовательности является существенным и список [1,2,3] не эквивалентен списку [3,2,1].

Элементами списка могут быть любые термы — константы, переменные, структуры, последние могут включать в себя другие списки.

В отличие от Лиспа, в Прологе списки — один из частных видов структур. Список — это либо пустой список, не содержащий не одного элемента, либо структура, имеющая два компонента : голову и хвост. Конец списка представляется как хвост, который является пустым списком. Для представления списка как частного вида структуры наиболее наглядна функторная форма записи, где голова и хвост списка являются компонентами функтора, обозначаемого точкой «.». К примеру, список [1,2,3] при использовании функторной формы записи представляется в виде следующей структуры :

.(1, .(2, .(3,[ ])))

3.1.2 Описание списков в Пролог-программе.

В Прологе для записи списков используется скобочная форма записи. Работа со списками основана на расщеплении на голову и хвост : [Head|Tail]. Голова есть первый аргумент функтора «.», хвост - второй : . (Head, [Tail]]). Функтор «.» используется для конструирования списка. Хвост списка есть список, состоящий из всех элементов исходного списка, за исключением первого.

Примеры :

[1,2,3]. Здесь Head : 1, Tail : [2,3]
[[1], [2]]. Здесь Head : [1], Tail : [[2]].

Для использования списка в программе необходимо описать предикат, который будет иметь список в качестве одного из своих аргументов. Кроме того, в разделе domains необходимо задать домен для этого списка.Сам список задается в программе либо в разделе clauses, либо в разделе goal.

Сопоставление списков в Турбо-Прологе и Visual Prolog'е ведется путем конкретизации входящих в них переменных в соответствии с правилами, изложенными в таблице 3.1.

Таблица 3.1. Правила сопоставления списков

Список 1	Список 2	Результат
[X,Y,Z]	[cat,dog,mouse]	X=cat Y=dog Z=mouse
[dog]	[X|Y]	X=dog Y=[ ]
[X,Y|Z]	[cat,dog,mouse]	X=cat Y=dog Z=[mouse]
[[b,c]|Z]	[[X,Y]|[w,b]]	X=b Y=c Z=[[w,b]]
[X,Y|Z,W]	Синтаксически некорректная конструкция списка	
[white,cat]	[cat,X]	Сопоставление невозможно

3.1.3 **Пример 1 — суммирование элементов списка.**

Кол-во аргументов : 1, тип аргумента : список целых чисел.

Условие завершения рекурсии : пустой список.

```
/* Суммирование элементов списка :
   листинг программы для реализации правила. */
domains
    t_elem=integer
    t_list=t_elem*
predicates
    sum_lst(t_list,t_elem).
clauses
    sum_lst([ ] ,0 ).
    sum_lst([Head|Tail], Sum):-
```

sum_lst(Tail, Sum1),
Sum = Head + Sum1 .

Этапы решения задачи «Суммирование элементов списка» на примере нахождения суммы элементов списка [1 , 2, 3, 4] представлены на рис.3.1.

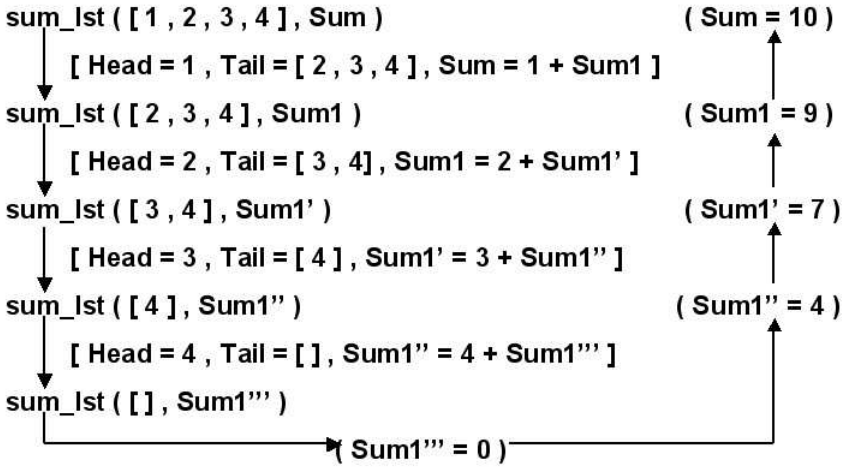

Рис. 3.1. Этапы решения задачи «Суммирование элементов списка»

3.1.4 **Пример 2 — принадлежность списку.**

Постановка задачи.

Дан объект и список элементов того же типа, что и объект. Требуется выяснить, присутствует ли заданный объект среди элементов списка.

Решение.

Количество аргументов : 2.

Условие окончания рекурсии — очередной элемент списка равен искомому объекту :

member(X, [X|_]).

В случае несовпадения искомого объекта и головы списка результирующее значение вычисляется путем обращения к предикату проектируемого правила с хвостом списка в качестве второго аргумента :

member(X, [H|T]):— not(H=X), member(X, T).

```
/* Принадлежность списку целых чисел :
   листинг программы для реализации правила. */
domains
    t_elem=integer
    t_list=t_elem*
```

```
predicates
    member(t_elem,t_list).
clauses
    member(X, [X|_]).
    member(X, [H|T]):— not(H=X), member(X, T).
```

3.1.5 **Пример 3 — объединение двух списков.**

Постановка задачи.

Даны два списка. Требуется построить список — результат присоединения одного списка к другому.

Решение.

Количество аргументов : 2.

Тип аргументов — списки элементов заданного типа (в приведенном здесь примере — списки целых чисел).

Условие выхода из рекурсии — пустой список-первый аргумент.

Результат формируется путем присоединения головы первого списка в качестве головы к результату вызова проектируемого правила с хвостом первого списка в качестве первого аргумента предиката проектируемого правила и вторым списком в качестве второго аргумента.

```
/* Объединение двух списков :
   листинг программы для реализации правила. */
domains
    t_elem=integer
    t_list=t_elem*
predicates
    append(t_list,t_list,t_list).
clauses
    append([ ], L, L).
    append([Head_lst1|Tail_lst1], Lst2, [Head_lst1|Tail_res]):—
          append(Tail_lst1, Lst2, Tail_res).
```

3.2 **Задание на лабораторную работу.**

Задание 1.

Написать программу сортировки списка методом Шелла. Вычисление последовательности шагов сортировки производится в соответствии с вариантом в Таблице 3.2.

Таблица 3.2. Вычисление шага сортировки Шелла

Вариант	Вычисление последовательности шагов
1 — 8	Методом Р. Седжвика : $h_k = \begin{cases} 9 * 2^k - 9 * 2^{k/2} + 1 & \text{если } k \text{ четно} \\ 8 * 2^k - 6 * 2^{(k+1)/2} + 1 & \text{если } k \text{ нечетно} \end{cases}$ При использовании формулы Седжвика следует остановиться на значении h_{k-1}, если $3 * h_k > n$, где n — длина списка.
9 — 16	Методом, предложенным Дональдом Кнутом : $h_{k-1} = 3 * h_k + 1$, $h_t = 1$, где h_k — шаг сортировки, t — число шагов сортировки : $t = [\log_3 n] - 1$, n — длина списка.
17 — 25	Методом, предложенным Дональдом Кнутом : $h_{k-1} = 2 * h_k + 1$, $h_t = 1$, где h_k — шаг сортировки, t — число шагов сортировки : $t = [\log_2 n] - 1$, n — длина списка.

Задание 2.

Написать программу сортировки списка в соответствии с вариантом в Таблице 3.3.

Таблица 3.3. Методы сортировки

Вариант	Реализуемый метод сортировки
1	Сортировка простыми включениями.
2	Сортировка бинарными включениями.
3	Сортировка методом прямого выбора.
4	Сортировка методом пузырька.
5	Шейкер-сортировка.
6	Сортировка Хоара.

Сравнить эффективность реализованной сортировки и реализованного в **Задании 1** варианта сортировки Шелла.

Задание 3.

Написать программу решения головоломки. Варианты заданий указаны в Таблице 3.4.

Таблица 3.4. Варианты заданий

Вариант	Задание
1	«Числа по периметру». По периметру пятиконечной звезды необходимо проставить числа от 1 до 10 так, чтобы суммы чисел в концах любого отрезка не делились ни на 3, ни на 5, ни на 7.
2	«Слово из домино». Необходимо сложить слово «игра», используя все 28 косточек домино.

Продолжение таблицы 3.4

Вариант	Задание
	При построении должны соблюдаться следующие условия : — Суммы очков косточек домино во всех четырех буквах должны быть одинаковы. — Косточки в каждой букве должны быть сложены одна к другой так, как это предусмотрено правилами игры в домино (косая перекладина из двух косточек в букве «и» также подчиняется этому правилу.).
3	«Пирамида из домино». Требуется расположить комплект домино в виде пирамиды, соблюдая следующие условия : — В каждой строчке сумма очков на косточках должна быть точным квадратом. — В строчках косточки укладываются согласно правилам игры в домино : 0 к 0, 1 к 1 и т. д.
4	«Звезда из домино». Требуется расположить все 28 косточек домино в виде семиконечной звезды (по четыре косточки на луче) так, чтобы: — В центр выходили кости с 0, 1, 2, 3, 4, 5, 6 очками; — На концах лучей также были все очки от 0 до 6; — В каждом луче косточки укладывались согласно правилам игры в домино : 0 к 0, 1 к 1, и т. д.; — Суммы очков на косточках домино во всех лучах были равными.
5	«Рамки из домино». Из 28 косточек домино требуется выложить четыре рамки так, чтобы сумма очков вдоль каждой стороны каждой рамки равнялась 13. Прикладывать косточки друг к другу одинаковыми значениями очков не обязательно.

Задание 4.

Написать программу обработки списка в соответствии с вариантом в Таблице 3.5.

Таблица 3.5. Варианты заданий

Вариант	Задание
1	Написать программу преобразования списка, которая реверсирует n элементов исходного списка, начиная с i-го.
2	Написать программу вставки подсписка в список, начиная с i-го элемента.

Продолжение таблицы 3.5

Вариант	Задание
3	Написать программу вставки в список нового элемента на все $(i * n)$-е места, где i задается в качестве аргумента, а $n = 1, 2, 3, \ldots$.
4	Написать программу удаления из списка элементов, находящихся на $(i * n)$-х местах, где i задается в качестве аргумента, а $n = 1, 2, 3, \ldots$.
5	Написать программу включения в исходный список элементов другого списка с заданным интервалом i.
6	Написать программу, подсчитывающую число вхождений некоторого подсписка в заданный список.
7	Написать программу, удаляющую все вхождения некоторого подсписка в исходный список.

3.3 **Содержание отчета по работе.**

Отчет по работе должен содержать :

1) Формулировку цели и задач проводимых исследований;
2) Для каждого задания — анализ задания, обзор методов решения, выбор метода решения с обоснованием, описание процесса разработки программ, полученные результаты и их анализ (обоснование);
3) Тексты программ для всех выполненных заданий с комментариями и обоснованиями;
4) Выводы по проведенным машинным экспериментам.

ЛАБОРАТОРНАЯ РАБОТА №4

ИСПОЛЬЗОВАНИЕ ОТСЕЧЕНИЙ

Цель работы.

Целью работы является изучение правил использования отсечения в двух случаях : для подтверждения правильности выбранного решения и для прекращения процесса порождения и проверки возможных решений.

4.1 Задание на лабораторную работу.

Задание 1.

Создать меню, обеспечить вызов и выполнение всех заданий из Лабораторной работы № 3. Каждое задание должно выполняться в отдельном окне. При выполнении работы в среде Visual Prolog рекомендуется использовать средства создания графического интерфейса (см. ниже, а также прилагаемый пример в директории **Demo_lab4_ver2**). Реализовать процедуру ввода списка. При использовании среды визуальной разработки Visual Prolog изучить принципы расстановки отсечений в предложениях, которые генерируются экспертом кода и обрабатывают события от пунктов меню.

Задание 2.

Написать для игры «Крестики-нолики» процедуру «Следующий ход», которая для заданного положения на доске находила бы наилучший ход, обеспечивающий либо предотвращение проигрыша, либо выигрыш, либо наилучший прогнозируемый результат.

Задание 3.

Написать программу, реализующую выигрышную стратегию для игры «Крестики-нолики» на доске 3×3. Игровое поле и весь процесс игры должен отображаться на экране в графическом режиме.

4.2 Краткие теоретические сведения.

4.2.1 Отсечение и его использование.

Причины введения отсечения.

Отсечение позволяет указать, какие из сделанных ранее выборов не следует пересматривать при возврате по цепочке согласованных Целевых Утверждений (ЦУ).

Преимущества введения отсечений :

— Сокращение времени выполнения программы за счет отсутствия новых сопоставлений для целей, не способных более внести ничего нового в решение;

— Уменьшение занимаемого программой объема памяти за счет отсутствия необходимости запоминания точек возврата.

Синтаксис, эффект и общие случаи использования отсечения.

Использование отсечения в правиле выглядит как вхождение ЦУ с предикатом «!».

Отсечение :

— не имеет аргументов;
— всегда согласуется с базой данных;
— не допускает повторного согласования;
— имеет побочный эффект, который изменяет процесс последующего возврата.

Изменение процесса последующего возврата заключается в недоступности маркеров некоторых целей. Примером может послужить работа с базой данных выдачи книг библиотекой.

```
/* Библиотека */
predicates
        reader(symbol).
        book_not_give(symbol,symbol).
        basic_service(symbol).
        addition_service(symbol).
        common_service(symbol).
        service(symbol,symbol).
clauses /* Данные о читателях */
        reader(«Иванов Иван Иванович»).
        reader(«Петров Петр Петрович»).
        reader(«Сидоров Сидор Сидорович»).
        reader(«Смирнов Алексей Анатольевич»).
        /* Данные о не возвращенных в срок книгах */
        book_not_give(«Иванов Иван Иванович»,
                      «Опыт теории лингвистических моделей»).
        book_not_give(«Петров Петр Петрович»,
                      «Лексическая семантика»).
        book_not_give(«Сидоров Сидор Сидорович»,
                      «Системы машинного перевода ЭТАП-2»).
        /* Данные об основных услугах */
        basic_service(«Пользование каталогом»).
        basic_service(«Справочно-библиографические услуги»).
        /* Данные о дополнительных услугах */
        addition_service(«Абонемент»).
        addition_service(«Межбибилиотечный абонемент»).
        /* Услуги, предоставляемые библиотекой */
```

```
common_service(X) :— basic_service(X).
common_service(X) :— addition_service(X).

/* Виды услуг, доступные читателю */

service(Reader,Service) :—
                    book_not_give(Reader,Book),
                    !,
                    basic_service(Service).
service(Reader,Service) :—
                    common_service(Service).
```

goal

```
reader(«Иванов Иван Иванович»),
service(«Иванов Иван Иванович», Service).
```

Отсечение в программе «отсекает» путь, представляющий цепочку выполненных доказательств, таким образом, что следующая цель соединяется непосредственно с исходной, рис.4.1.

Рис. 4.1. Диаграмма согласования целевого утверждения при наличии отсечения

В приведенном на рис.4.1 примере результат действия отсечения в правиле для предиката *service* заключается в том, что все цели, выбранные с момента, когда было выбрано это правило, запоминаются в системе как неизменяемые при обратном просмотре. Целевое утверждение *service* называется родительским целевым утверждением для отсечения, поскольку именно это целевое утверждение привело к использованию правила, содержащего отсечение.

Если отсечение встречается в качестве целевого утверждения, то после этого система лишается возможности изменять решения, принятые ею с момента вызова родительского утверждения. Все альтернативы принятым решениям отбрасываются. Попытка вновь доказать согласованность с базой данных любого целевого утверждения между родительским целевым утверждением и отсечением заканчивается неудачей.

Общие случаи использования отсечения :

— Указание Прологу на то, что найдено нужное правило для заданного целевого утверждения.
— Указание Прологу необходимости немедленного прекращения доказательства согласованности конкретного целевого утверждения без поиска альтернативных решений. В этом случае используется конъюнкция отсечения с предикатом fail, что означает : «если вы дошли до этого места, то вам следует прекратить попытки доказать согласованность данного целевого утверждения».
— Завершение «порождения и проверки». Использование механизма возврата для прекращения порождения альтернативных решений. В этом случае отсечение означает : «если вы дошли до этого места, то вы нашли единственное решение задачи и никакой возможности найти другие альтернативные решения нет».

Примером использования отсечения для подтверждения правильности выбора правила может послужить нахождение суммы целых чисел от 1 до N :

```
predicates
        sum(integer,integer).

clauses

        sum(N,1):-N<=1,!.
        sum(N,R):-N1=N-1,sum(N1,R1),R=R1+N.
```

Использование механизма отсечения для указания Прологу на ситуации, когда он выбрал единственно правильное правило, может быть заменено использованием предиката not. not(X) означает, что X недоказуемо как целевое утверждение Пролога. Так, в примере для суммы целых чисел от 1 до N будем иметь :

```
predicates
        sum(integer,integer).

clauses

        sum(N,1):-N<=1.
        sum(N,R):-not(N<=1),N1=N-1,sum(N1,R1),R=N+R1.
```

Примером использования конъюнкции отсечения и fail может послужить программа вычисления размера налога на основе правила «средний налогоплательщик» :

```
predicates
        wife(symbol,symbol).
        mid_taxp(symbol).
        foreigner(symbol).
        income(symbol,integer).
        salary(symbol,integer).
        cap_income(symbol,integer).
        received_grant(symbol,integer)
```

```
clauses
        wife(mary,ivan).
        foreigner(bill).
        salary(mary,120).
        salary(ivan,200).
        salary(sergey,150).
        salary(oleg,100).
        cap_income(bill,150).
        cap_income(sergey,200).
        received_grant(peter,50).

        /* Средний налогоплательщик */

        mid_taxp(X) :- foreigner(X), !, fail.
        mid_taxp(X) :-
                        wife(X,Y), income(Y,Income),
                        Income>150, !, fail.
        mid_taxp(X) :-
                        income(X,Income), Income>=50,
                        Income<=150.
        income(X,Y) :-
                        received_grant(X,Y), Y<50, !, fail.
        income(X,Y) :- salary(X,Y).
        income(X,Y) :- salary(X,Z),
                        cap_income(X,W), Y=Z+W.
```

Суть использования отсечения в приведенном примере заключается в том, что попытка доказать, что некоторый человек является средним налогоплательщиком, может быть прервана при выполнении некоторого условия (или группы условий). Комбинация «Отсечение+fail» дает возможность остановить поиск альтернатив, прежде чем будет выполнен предикат fail. «Отсечение+fail» может быть заменено использованием not :

```
        not_midt(X) :- wife(X,Y), income(Y,Income), Income>150.
        mid_taxp(X) :- not(foreigner(X)), not(not_midt(X)),
                          income(X,Income), Income>=50, Income<=150.
        income(X,Y) :- received_grant(X,Y), not(Y<50).
        income(X,Y) :- salary(X,Y).
        income(X,Y) :- cap_income(X,Y).
        income(X,Y) :- salary(X,Z), cap_income(X,W), Y=Z+W.
```

Примером использования отсечения для завершения «порождения и проверки» может послужить реализация игры в «крестики-нолики». Введение отсечения здесь равносильно следующему заявлению : «если идет поиск вынужденных ходов, то важно найти только первое решение».

```
        /* Игровое поле, рис.4.2 */

        field_line([1,2,3]).
        field_line([4,5,6]).
        field_line([7,8,9]).
```

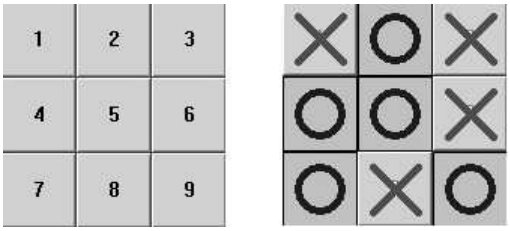

Рис. 4.2. Реализация игрового поля «крестики-нолики»

```
field_line([1,4,7]).
field_line([2,5,8]).
field_line([3,6,9]).
field_line([1,5,9]).
field_line([3,5,7]).
/* Символ на позиции с заданным номером */
arg(_,[],«empty»).
arg(1,[H_board|T_board],H_board):-!.
arg(Pos,[_|T_board],Sym):-
                Pos1=Pos-1,arg(Pos1,T_board,Sym).
cross_pos(Pos,Board):-
                arg(Pos,Board,Sym),Sym=«X».
zero_pos(Pos,Board):-
                arg(Pos,Board,Sym),Sym="0".
empty_pos(Pos,Board):-
                arg(Pos,Board,Sym),Sym="empty".
thread([Pos,X,Y],Board,Pos):-
                empty_pos(Pos,Board),
                cross_pos(X,Board),
                cross_pos(Y,Board).
thread([X,Pos,Y],Board,Pos):-
                empty_pos(Pos,Board),
                cross_pos(X,Board),
                cross_pos(Y,Board).
thread([X,Y,Pos],Board,Pos):-
                empty_pos(Pos,Board),
                cross_pos(X,Board),
                cross_pos(Y,Board).
thread([Pos,X,Y],Board,Pos):-
                empty_pos(Pos,Board),
                zero_pos(X,Board),
                zero_pos(Y,Board).
thread([X,Pos,Y],Board,Pos):-
                empty_pos(Pos,Board),
                zero_pos(X,Board),
                zero_pos(Y,Board).
thread([X,Y,Pos],Board,Pos):-
                empty_pos(Pos,Board),
```

```
                zero_pos(X,Board),
                zero_pos(Y,Board).
/* Вынужденный ход */
forced_move(Board,Pos):-
                field_line(Field_line),
                thread(Field_line,Board,Pos),!.
```

Другой пример управления «порождением и проверкой» — реализация целочисленного деления :

```
predicates
        int_num(integer).
        div(integer,integer,integer).
clauses /* Генератор целых чисел */
        int_num(0).
        int_num(X):-int_num(Y),X=Y+1.
        /* Предикат целочисленного деления */
        div(N1,N2,Result):-
                int_num(Result),
                P1=Result*N2,
                P2=(Result+1)*N2,
                P1<=N1,P2>N1,!.
```

Правило div использует предикат int_num как генератор всех возможных целых чисел-альтернатив, остальные целевые утверждения выполняют функцию контроллеров. Если убрать отсечение, то любой возврат будет заново инициализировать поиск альтернатив для int_num. При этом ни одно значение не было бы правильным результатом деления и генерация целых чисел продолжалась бы до бесконечности.

Проблемы, связанные с использованием отсечения.

Если отсечение вводится с целью обеспечения правильности работы программы, то нет гарантии, что появление целевых утверждений иной формы не приведет к непредвиденному результату. Пример — измененное определение предиката append :

```
/* Исходный вариант */
append([ ],L,L).
append([H1|T1],L2,[H1|T]):-append(T1,L2,T).
/* Измененный вариант */
append1([ ],L,L):-!.
append1([H1|T1],L2,[H1|T]):-append1(T1,L2,T).
```

Рассмотрим поведение правил append и append1 в случае использования незапланированным способом. Пусть имеются ЦУ append(X,Y,[1,2,3]) и append1(X,Y,[1,2,3]). По аналогии с первым второе ЦУ будет сопоставлено с заголовком первого правила для append1, что даст

X=[ ],Y=[1,2,3], но у append1 затем встретится отсечение. Это приводит к тому, что альтернативные варианты append1 будут недоступны, хотя для данного запроса у append имеются другие решения : X=[1], Y=[2,3] ; X=[1,2], Y=[3] ; X=[1,2,3], Y=[ ].

Таким образом, надежное использование отсечения возможно лишь в случае наличия полной и достоверной информации о характере использования правил. Если характер использования правил меняется, то необходимо пересмотреть все случаи употребления отсечения.

4.2.2 Создание графического пользовательского интерфейса в Visual Prolog.

Подробное описание среды визуальной разработки Visual Prolog содержится в [1, с. 633–807].

Рассмотрим основные предикаты VPI (Visual Programming Interface, интерфейс визуального программирования).

Запуск VPI-приложения начинается с выполнения секции GOAL, из которой вызывается предикат vpi_Init (см. распечатку файла test.pro из примера в Приложении А) :

```
vpi_Init(task_win_Flags,task_win_eh,task_win_Menu,
                «test»,task_win_Title).
```

Этот предикат запускает систему управления окнами и создает главное окно приложения (**Task Window**). Код, предшествующий вызову предиката vpi_Init, — это определение параметров предиката vpi_Init.

Эксперт приложений [1, с. 633] создаст раздел GOAL со всеми атрибутами и флагами, которые нами устанавливаются вручную в диалоговом окне **Application Expert**. Любые иные действия по инициализации приложения выполняются в обработчике события e_create, которое посылается окну **Task Window**. Все действия по завершению приложения должны быть выполнены в обработчике события e_destroy. Предикат vpi_Init закончит свою работу только после того, как будет закрыто окно **Task Window**.

Каждое окно или диалоговое окно в приложении представлены структурой памяти и предикатом обработчика событий окна, которому VPI посылает сообщения при возникновении пользовательских или системных событий, связанных с окном. Приложение может посылать событие любому из своих обработчиков событий. В VPI названия событий из домена событий всегда имеют префикс e_.

Создание окон.

Создание окна выполняется предикатом, имя которого начинается с «win_», и оканчивается на «_Create». В прилагаемом примере окно с заголовком «Кнопки для реализации игры» будет создано при вызове предиката win_Create (см. распечатку файла crosses_zeros_demo.rpo в Приложении А).

Данный предикат имеет следующий синтаксис :

win_Create(Windowtype,Rct,Title,Menu,ParentWindow,
WSFlags,EventHandler,CreationData).

Здесь тип Windowtype определяет вид окна с заголовком Title и рабочей областью Rct (Rct имеет тип rct(integer,integer,integer,integer) и задает координаты верхнего левого и правого нижнего угла). В VPI окна классифицируются по типам [1, с. 704]. Типы окон должны принадлежать домену WINDOWTYPE и иметь префиксы w_ (для окон), wc_ (для элементов управления) или wd_ (для диалоговых окон). Типы окон, используемые в примере crosses_zeros_demo.pro из Приложения А, перечислены в Таблице 4.1.

Таблица 4.1. Типы окон в примере из Приложения А

Тип окна	Разделитель
w_TopLevel	Обычное окно документа
w_Task	Окно Task Window
wc_PushButton	Командная кнопка
wc_Text	статический текст

Для определения стиля окна или диалогового окна используются флаги [1, с. 705]. Все флаги списка WSFlags имеют префикс wsf_. Флаги могут быть определены в теле программы как непосредственным заданием, так и автоматически при создании окна посредством редактора окон (диалоговых окон). При этом [1, с. 649] перед запуском такого редактора появляется диалоговое окно **Window Attributes** (или, соответственно, **Dialog Attributes**), где устанавливаются флажки того или иного стиля создаваемого окна (диалогового окна). Сам код для окна (диалогового окна) генерируется с помощью эксперта окон и диалоговых окон [1, с. 650].

В прилагаемом примере используются следующие стилевые флаги для окна «Кнопки для реализации игры» :

— wsf_TitleBar — окно получает строку заголовка;
— wsf_Border — окно получает тонкую границу;
— wsf_Close — окно получает системное меню, которое обычно используется для закрытия окна.

Для определения того, какое меню должно иметь окно, используется 4-й аргумент Menu предиката win_Create типа MENU (более подробно о домене MENU см.йнтерактивную справку Visual Prolog'a). В прилагаемом примере окно «Кнопки для реализации игры» не имеет меню и значение данного аргумента равно no_menu.

5-й аргумент ParentWindow предиката win_Create содержит дескриптор родительского окна. EventHandler, 7-й аргумент, EventHandler, определяет предикат, который вызывается при поступлении события для окна. В прилагаемом примере это предикат win_кнопки_для_реализации_игры_eh. Он описан в файле crosses_zeros_demo.pro.

8-й аргумент CreationData предиката win_Create есть значение типа long, которое будет передано обработчику событий как часть события

e_create.

Создание диалоговых окон.

Диалоговые окна [1, с. 708] — это специальный вид окон. Их функциональные возможности ограничены по сравнению с обыкновенными окнами. Диалоговые окна обычно содержат элементы управления, через которые происходит взаимодействие приложения с пользователем (отображение выходной информации, ввод входных данных, выбор из множества свойств или редактирование). Операционная система позволяет перемещаться между элементами управления при помощи клавиши <Tab> и комбинации клавиш <Shift>+<Tab>.

Для создания диалогового окна используется предикат win_CreateResDialog. Данный предикат имеет следующий синтаксис:

win_CreateResDialog(ParentWindow,Windowtype,ResId,
EventHandler,CreationData),

Здесь ResId — беззнаковое целое, представляющее собой идентификатор ресурса.

В зависимости от значения параметра Windowtype создается либо модальное, либо немодальное окно. Модальное диалоговое окно должно быть закрыто прежде, чем пользователь сможет активизировать любое другое окно, меню или элемент управления в приложении.

Для облегчения разработки и отладки приложений в Visual Prolog существует ряд стандартных диалоговых окон. Они создаются непосредственно приложениями при помощи вызова одного предиката. Вставку такого вызова в нужное место текста программы можно осуществить из подменю **Edit** главного меню среды Visual Prolog, либо правой кнопкой «мыши» : **Insert|Predicate call|Window, Dialog or Toolbar**. В прилагаемом примере используются стандартные диаговые окна : ввода строки (предикат dlg_GetStr), выбора элемента из списка значений (предикат dlg_ListSelect), примечания (предикат dlg_Note).

Создание меню.

Среда Visual Prolog'a позволяет создавать меню в автоматическом режиме. Для создания и редактирования меню используется редактор меню [1, с. 657, 785]. При использовании режима MDI (Multiple Document Interface, соответствующий флаг должен быть установлен в **Application Expert**) все меню будут появляться в окне **Task**.

Ресурс создаваемого меню описывается с помощью домена MENU. Ссылка на домен, который используется при создании конкретного меню, делается в разделе **constants**. Эта константа используется в качестве значения 3-го аргумента предиката vpi_Init при запуске системы управления окнами в разделе GOAL (см. распечатку файла test.pro).

Для определения действия над пунктом меню в обработчике событий окна, для которого создается меню, с помощью эксперта кода

генерируется предложение, которое обрабатывает событие от этого пункта меню.

К примеру, для пункта **Демо-кнопки** в меню **Task Menu** прилагаемого примера событие обрабатывается следующим кодом :

```
task_win_eh(_Win,e_Menu(id_демокнопки,_ShiftCtlAlt),0):-!,
        win_кнопки_для_реализации_игры_Create(_Win),!.
```

Обработчики событий.

С каждым окном и диалоговым окном связывается предикат, который обрабатывает все события для этого окна. Он принадлежит домену EHANDLER [1, с. 709], имеет два входных параметра : дескриптор окна и событие. Предикат возвращает параметр типа LONG, значение которого зависит от обрабатываемого события.

События для окна «Кнопки для реализации игры» в прилагаемом примере обрабатываются предикатом win_кнопки_для_реализации_игры_eh (см. распечатку файла cross-es_zeros_demo.pro). Обрабатываемые события для окна «Кнопки для реализации игры» относятся к доменам e_create, e_control, e_size и e_menu.

Событие e_size сообщает, что окно изменило размер [1, с. 725].

Элементы управления посылают своему родительскому окну события e_control [1, с. 727] :

```
e_control(CtlId,CtlType,CtlWindow,Control_info).
```

Здесь :

CtlId — целочисленная константа, которая уникально идентифицирует элемент управления в окне или диалоговом окне.

CtlType — одна из констант с префиксом wc_ (см. Таблицу 4.1), задает тип элемента управления. Для окна «Кнопки для реализации игры» это тип wc_PushButton.

CtlWindow — дескриптор окна элемента управления;

Control_info — это вспомогательное сообщение, которое более подробно объясняет обработчику событий причину активизации.

Когда пользователь выбирает пункт меню, приложению посылается событие e_menu [1, с. 721] :

```
e_menu(MenuTag,ShiftControlAlt)
```

Здесь :

MenuTag — идентификатор константы, которая определяет выбранный пункт меню.

ShiftControlAlt — целочисленный параметр, который содержит состояния клавиш <Shift> и <Ctrl> во время активизации пункта меню.

Установка активного окна.

Осуществляется вызовом предиката win_SetActiveWindow с дескриптором WIN1 активизируемого окна в качестве аргумента, после чего необходимо изменить фокус ввода на активизируемое окно посредством вызова предиката win_SetFocus :

```
win_SetActiveWindow(WIN1),
win_SetFocus(WIN1).
```

Обновление и уничтожение окон.

Вызов предиката win_Update(WIN1) инициализирует обновление клиентской части окна с дескриптором WIN1 посредством отправки сообщения e_Update обработчику событий. Вызов предиката win_Destroy уничтожает окно, диалоговое окно или элемент управления. При этом обработчик событий окна получает событие e_Destroy. При уничтожении окна автоматически будут уничтожены и все его дочерние окна. При этом дочерние окна получают событие e_Destroy перед тем, как его получит родительское окно.

4.3 Содержание отчета по работе.

Отчет по работе должен содержать :

1) Формулировку цели и задач проводимых исследований;
2) Для каждого задания — анализ задания, обзор методов решения, выбор метода решения с обоснованием, описание процесса разработки программ, полученные результаты и их анализ (обоснование);
3) Тексты программ для всех выполненных заданий с комментариями и обоснованиями;
4) Для задания 3 — описание выигрышной стратегии;
5) Выводы по проведенным машинным экспериментам.

ЛАБОРАТОРНАЯ РАБОТА №5

РЕШЕНИЕ ЛОГИЧЕСКИХ ЗАДАЧ МЕТОДОМ ПОИСКА НА ПРОСТРАНСТВЕ СОСТОЯНИЙ

Цель работы.

Целью работы является овладение методологией решения логических задач с применением известных на сегодняшний день стратегий поиска в пространстве состояний.

5.1 Задание на лабораторную работу.

Задание 1.

Изучить на приведенном ниже примере задачи о волке, козе и капусте работу базовой программы для решения задач методом поиска в глубину.

Задание 2.

Написать программу для решения задачи о двух кувшинах.

Формулировка задачи :

Дан кувшин с водой емкостью N и пустой кувшин емкостью M. Требуется получить заданную емкость L. Воду можно либо выливать, либо переливать из одного кувшина в другой. (Кувшины можно полностью наполнять водой из неограниченного резервуара).

Программа должна сохранять все состояния задачи в памяти, позволять просматривать эти состояния. Вызов задачи и просмотр выполнять из меню.

Задание 3.

Решить задачу в соответствии с вариантом в Таблице 5.1.

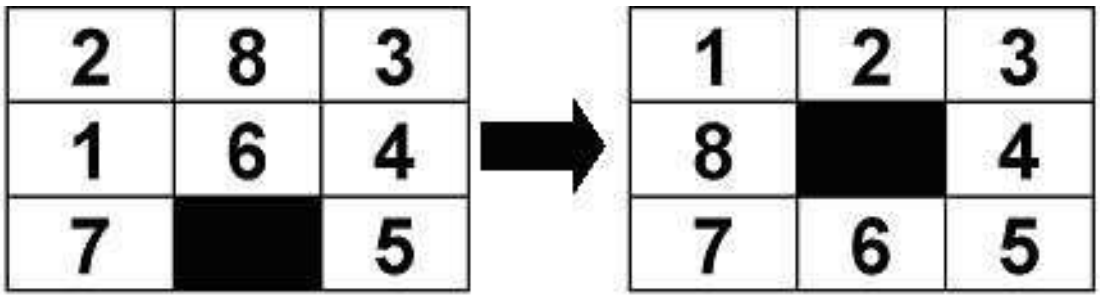

Рис. 5.1. Пример начальной и целевая конфигурация фишек для игры в восемь

Таблица 5.1. Варианты заданий

Вариант	Условие задачи
1	Игра в восемь. На игровом поле 3×3 расставлены фишки с цифрами от 1 до 8. Имеется одна свободная клетка, на которую могут быть передвинуты фишки с соседних позиций. При этом не допускается перемещение фишек через одну, а также по диагонали. Задача состоит в том, чтобы преобразовать некоторую начальную конфигурацию в целевую конфигурацию, рис.5.1.
2	Задача о миссионерах и каннибалах. Три миссионера и три каннибала находятся на левом берегу реки. Все хотят перебраться на другой берег. Здесь же небольшая лодка, вмещающая не более двух человек. Если на каком-то берегу каннибалов окажется больше, чем миссионеров, то они съедят миссионеров. Если окажется больше миссионеров, то они обратят каннибалов в свою веру. Найти последовательность ездок, гарантирующую безопасность миссионерам и свободу вероисповедания каннибалам.
3	Задача о шахматном коне (задача Эйлера). Требуется обойти все клетки шахматной доски ходом коня.
4	Путь коня. На шахматной доске N×N, несколько клеток из которой вырезано, заданы две клетки. Построить минимальный путь коня из одной клетки в другую.
5	Магараджи и пери. Случилось так, что к берегу великого Ганга подъехали сразу трое магараджей со своими пери. Все они хотели переправиться на другой берег, но обычаи не позволяли ни одной пери оставаться в лодке или на берегу одной с чужим мужем, если рядом не будет своего. У берега стояла небольшая лодка, которая выдерживала только двух человек. Обычаи не запрещают пери самой управлять лодкой. Требуется найти последовательность перемещений лодки с одного берега на другой, гарантирующую перемещение всех магараджи и пери на другой берег.

Продолжение таблицы 5.1

Вариант	Условие задачи
6	Маршрут слона. На шахматной доске стоит слон. Требуется найти маршрут обхода слоном всех клеток заданного цвета на шахматной доске.
7	Расстановка жирафов. Новая шахматная фигура «жираф» ходит буквой «Г» на четыре клетки в одном направлении и на пять клеток — в другом. Требуется написать программу, которая находила бы для стандартной шахматной доски расстановку максимального числа жирафов таким образом, чтобы ни один из расставленных жирафов не мог напасть на другого, сколько бы он ни ходил.

5.2 Краткие теоретические сведения.

5.2.1 Состояния и операторы.

Определение 5.2.1. *Под состоянием задачи будем понимать некоторая конфигурация исследуемой динамически меняющейся системы. При этом начальную и целевую конфигурации принято рассматривать в качестве, соответственно, начального и целевого состояний.*

Определение 5.2.2. *Оператор преобразует одно состояние в другое. В общем случае мы будем предполагать, что операторы — это вычисления, преобразующие одни описания состояний в другие.*

При строковом описании состояний удобным способом задания операторов являются правила переписывания или продукции.

Определение 5.2.3. *Правила переписывания (продукции) определяют возможные способы преобразования одной строки в другую. Правила переписывания задаются в форме* $S_i \to S_j$.

Определение 5.2.4. *Пространство состояний, достижимых из начального состояния, состоит из тех конфигураций системы, которые могут быть образованы из начальной с применением преобразований, допустимых некоторой совокупностью правил. Очень часто пространство состояний представляют в виде графа, вершины которого соответствуют состояниям, а дуги — операторам.*

При решении задач с использованием пространства состояний необходимо выбрать форму описания состояний задачи, эта форма должна быть единой для всех возможных состояний. Как правило, выбираемая форма описания имеет сходство с некоторым физическим свойством решаемой задачи и может принадлежать любому допустимому типу данных (массивы, строки, списки, деревья).

При формулировке задачи в пространстве состояний решение получается в результате применения операторов к описаниям состояний до

тех пор, пока будет получено выражение, описывающее целевое состояние. Решение задачи состоит в поиске пути из начального состояния в целевое посредством применения последовательности операторов. Эффективность механизма поиска может быть значительно повышена привлечением знаний проблемной области в виде разного рода эвристической информации. К примеру, для игры в восемь это может быть число фишек, которые лежат не на своем месте в описании текущего состояния.

Исследование различных стратегий перебора может быть весьма эффективно реализовано с применением языка теории графов.

5.2.2 **Поиск на графе.**

Все методы перебора в пространстве состояний могут быть смоделированы с помощью следующего теоретико-графового процесса :

1) Начальная вершина S соответствует описанию начального состояния. Вершины, непосредственно следующие за данной, получаются в результате использования операторов, которые применимы к описанию состояния, ассоциированного с этой вершиной.
2) Пусть γ — некоторый специальный оператор, который строит все вершины, непосредственно следующие за данной. Будем называть процесс применения оператора γ к вершине раскрытием вершины. Для каждой раскрываемой вершины делается проверка, является ли вершина целевой.
3) От каждой дочерней вершины к родительской идут указатели, которые позволяют найти путь назад к начальной вершине после обнаружения целевой вершины. Вершины и указатели, построенные в процессе перебора, образуют поддерево всего неявно определенного дерева пространства состояний. Данное поддерево называют деревом перебора.
4) Решающую последовательность образуют операторы, которые связаны с дугами пути от целевой вершины к начальной.

5.2.3 **Метод перебора в глубину.**

Определение 5.2.5. *Процесс перебора в глубину (depth-first process) характеризуется тем, что вначале раскрывается та вершина, которая была построена самой последней.*

Определение 5.2.6. *(глубина вершины в дереве)*

1) Глубина корня дерева равна нулю.
2) Глубина любой последующей вершины дерева равна глубине родительской вершине, увеличенной на 1.

Определение 5.2.7. *Вершиной с наибольшей глубиной в дереве перебора а некоторый момент служит раскрываемая в данный момент.*

Определение 5.2.8. *Граничная глубина при использовании метода перебора в глубину вводится в целях отсечения заведомо бесполезных путей в дереве и организации процедуры возврата. При построении вершины с глубиной, превышающей граничную, следующей будет раскрыта вершина наибольшей глубины, не превышающей граничного значения.*

5.2.4 **Базовая программа для решения задач поиска на графах состояний.**

Задаваемые операторами переходы будем описывать бинарным предикатом move(State, Move), где Move — правило перехода, применяемое к состоянию State.

Предикат update(State, Move, State1) используется для поиска состояния State1, достижимого с помощью применения правила Move к состоянию State. В ряде случаев целесообразным является объединение процедур move и update.

Допустимость возможных переходов оценивается предикатом legal(State), который проверяет, удовлетворяет ли состояние State ограничениям задачи.

В целях предупреждения зацикливания программа сохраняет ранее пройденные состояния. Последовательность переходов из начального состояния в целевое строится путем наращивания списка Moves в третьем аргументе правила solve_dfs :

solve_dfs(State, History, Moves),

где Moves — последовательность переходов до достижения требуемого конечного состояния из текущего состояния State. History содержит ранее пройденные состояния.

Приведем листинг базовой программы организации поиска в глубину.

```
/* Построение последовательности переходов из начального состояния
   в конечное.

   Первый аргумент — текущее состояние,
   Второй аргумент — список ранее пройденных состояний,
   Третий аргумент — последовательность переходов до достижения
   требуемого конечного состояния из текущего состояния */

   solve_dfs(State, History, [ ]): —
      final_state(State).

   solve_dfs(State, History, [Move|Moves]): —
      move(State, Move),
      update(State, Move, State1),
      legal(State1),
      not(member(State1, History)),
      solve_dfs(State1, [State1|History], Moves).

/* Запуск базовой программы для решения задач
   путем поиска на графах пространства состояний
   с применением поиска в глубину */

   test_dfs(Problem, Moves): —
      initial_state(Problem, State),
      solve_dfs(State, [State], Moves).
```

Замечание. *Чтобы использовать базовую программу организации поиска при решении задачи, программист должен выбрать формальное представление для состояний задачи и аксиоматизировать процедуры move, update и legal.*

5.2.5 Пример использования базовой программы организации поиска в глубину для решения задачи о волке, козе и капусте.

Состояния представляются тройками wgc(B,L,R), где B — местонахождение лодки (и фермера) — левый или правый берег, L — список находящихся на левом берегу, R — список находящихся на правом берегу.

Начальное состояние :

wgc(«Лодка на левом берегу.», [«Волк»,«Коза»,«Капуста»], [ ]).

Конечное состояние :

wgc(«Лодка на правом берегу.», [ ], [«Волк»,«Коза»,«Капуста»]).

Для проверки зацикливания удобно сохранять списки обитателей в отсортированном виде. При нахождении волка, козы и капусты на одном берегу волк будет в списке перед козой, и оба они — перед капустой.

Переходы из состояния в состояние — это перевозка обитателей с одного берега на другой, каждый переход может быть специфицирован конкретным грузом (Cargo).

Допустимость переходов определяется условием задачи : в отсутствие фермера волк и коза, равно как и коза с капустой, не могут в отсутствие фермера находиться на одном берегу.

Как показано в приведенном далее листинге программы для решения задачи о волке, козе и капусте, недетерминированное поведение предиката member позволяет компактно описать все возможные переходы тремя образцами правила move : для перевозки с левого берега на правый, для перевозки с правого берега на левый и для одиночного («Без груза») перемещения фермера на лодке с любого берега на любой.

```
/* Программа для решения задачи о волке, козе и капусте.
   Начальное и конечное состояние */
    initial_state(«Исходное состояние : все находятся на левом берегу »,
                  wgc(«Лодка на левом берегу.»,
                  [«Волк»,«Коза»,«Капуста»], [ ])).
    final_state(wgc(«Лодка на правом берегу.», [ ],
                  [«Волк»,«Коза»,«Капуста»])).
/* Определение возможности перехода из состояния в состояние */
    move(wgc(«Лодка на левом берегу.»,L,R), Cargo):—
      member(Cargo,L).
    move(wgc(«Лодка на правом берегу.»,L,R), Cargo):—
```

```
    member(Cargo,R).
  move(wgc(B,L,R), «Без груза»).
/* Поиск состояния, достижимого из заданного */
  update(wgc(B,L,R), Cargo, wgc(B1,L1,R1)):—
    update_boat(B,B1),
    update_banks(Cargo,B,L,R,L1,R1).
/* Изменение местонахождения лодки */
  update_boat(«Лодка на левом берегу.», «Лодка на правом берегу.»).
  update_boat(«Лодка на правом берегу.», «Лодка на левом берегу.»).
/* Выбор груза для перевозки */
  select(X,[X|T],T).
  select(X,[H|T],[H|D]):—
    select(X,T,D).
/* Порядок сортировки */
  precedes(«Волк»,X).
  precedes(X,«Капуста»).
/* Вставка элемента в список */
  insert(X,[Y|Ys],[X,Y|Ys]):—
    precedes(X,Y).
  insert(X,[Y|Ys],[Y|Zs]):—
    precedes(Y,X),
    insert(X,Ys,Zs).
  insert(X,[ ],[X]).
/* Изменение состава обитателей берегов
  Первый аргумент — перевозимый в лодке груз,
  Второй аргумент — текущее местонахождение лодки
                    (левый или правый берег),
  Третий и четвертый аргументы — текущий состав обитателей
                    правого и левого берега,
  Пятый и шестой аргументы — новый состав обитателей
                    правого и левого берега. */
/* Ситуация, когда фермер переправляется через реку без груза */
  update_banks(«Без груза»,B,L,R,L,R).
/* Ситуация перевозки обитателя одного берега на другой */
  update_banks(Cargo,«Лодка на левом берегу.»,L,R,L1,R1):—
    select(Cargo,L,L1),
    insert(Cargo,R,R1).
  update_banks(Cargo,«Лодка на правом берегу.»,L,R,L1,R1):—
```

```
    select(Cargo,R,R1),
    insert(Cargo,L,L1).
/* Допустимость состояния */
  legal(wgc(«Лодка на левом берегу.»,L,R)):—
    not(illegal(R)).
  legal(wgc(«Лодка на правом берегу.»,L,R)):—
    not(illegal(L)).
  illegal(List):—
    member(«Волк»,List),
    member(«Коза»,List).
  illegal(List):—
    member(«Коза»,List),
    member(«Капуста»,List).
```

5.3 **Содержание отчета по работе.**

Отчет по работе должен содержать :

1) Формулировку цели и задач проводимых исследований;
2) Для каждого задания :
   - Обзор методов решения задачи, выбор метода решения с обоснованием.
   - Описание представления в пространстве состояний (указать форму описания состояний, операторов, критериев достижения цели).
   - Представление пространства состояний в виде графа.
   - Описание процесса разработки программ.
   - Тексты программ с комментариями и обоснованиями.
   - Полученные результаты и их анализ (обоснование).
3) Выводы по проведенным машинным экспериментам.

ЛАБОРАТОРНАЯ РАБОТА №6

РЕШЕНИЕ ЛОГИЧЕСКИХ ЗАДАЧ МЕТОДОМ «ОБРАЗУЙ И ПРОВЕРЬ»

Цель работы.

Целью работы является изучение приемов использования метола «Образуй и проверь» при решении задач на Прологе.

6.1 Задание на лабораторную работу.

Решить задачу в соответствии с вариантом в Таблице 6.1. Задача обязательно должна быть решена методом «Образуй и проверь».

Таблица 6.1. Варианты заданий

Вариант	Условие задачи
1	Четверо ребят — Алексей, Борис, Иван и Григорий — соревновались в беге. На следующий день на вопрос, кто какое место занял, они ответили так : Алексей : «Я не был ни первым, ни последним». Борис : «Я не был последним». Иван : «Я был первым». Григорий : «Я был последним». Известно, что три из этих ответов правильные, а один — неверный. Требуется написать программу, которая определила бы, кто сказал неправду и кто был первым на самом деле.
2	Три купчихи — Олимпиада, Сосипатра и Поликсена — пили чай. Если бы Олимпиада выпила на 5 чашек больше, то она выпила бы столько, сколько две другие вместе. Если бы Сосипатра выпила бы на 9 чашек больше, то она выпила бы столько, сколько две другие вместе. Требуется написать программу, которая определяла бы, сколько каждая из трех купчих выпила чашек и кого из них какое отчество, если известно, что : — Уваровна пила чай вприкуску; — количество чашек чая, выпитых Титовной, кратно трем; — Карповна выпила 11 чашек.
3	В 9 «Г» классе учатся три брата : Алексей, Леонид и Александр. Учитель заметил, что : — если кто-то из них получает подряд две четверки или две тройки, то дальше он учится кое-как и получает тройку;

Продолжение таблицы 6.1

Вариант	Условие задачи
	— если он получает подряд две пятерки, то совсем перестает заниматься и получает двойку; — если он получает две разные оценки, то следующей будет большая из них. В начале полугодия Алексей получил оценки 4 и 5, Леонид — 3 и 2, Александр — 2 и 4. Требуется написать программу, которая определяла бы, какие итоговые оценки получит каждый из рассматриваемых школьников за данное полугодие, если учитель выставил каждому по 30 оценок, а итоговая оценка — ближайшее целое число к среднему арифметическому полученных оценок ?
4	Идет расследование преступления. Трое подозреваемых : Иванов, Петров и Сидоров дают показания. Чтобы окончательно запутать следствие, каждый из трех подозреваемых называет правильно либо марку, либо цвет машины, на которой преступники скрылись с места преступления. Показания подозреваемых : Иванов сказал, что машина — синие «Жигули». Петров утверждал, что была черная «Волга». Сидоров показал, что был «Мерседес», при этом отрицая, что цвет машины — синий. Требуется написать программу, которая определила бы марку и цвет машины, на которой преступники скрылись с места преступления.
5	Идет расследование преступления. Трое подозреваемых : Иванов, Петров и Сидоров дают показания : Иванов : «Я этого не делал. Это — Сидоров». Петров : «Сидоров этого не совершал. Это — Иванов». Сидоров : «Ни я, ни Петров этого не совершали». Следствием было установлено, что один из подозреваемых оба показания дал верно, другой — оба неверно, а третий — одно показание дал верно, другое — неверно. Требуется написать программу, которая устанавливала бы виновника преступления.
6	Есть пять коробочек: белая, черная, синяя, красная и зеленая. Шарики тех же цветов, что и коробочки, по два шарика каждого цвета. В каждой коробочке по два шарика каждого цвета. В каждой коробочке по два шарика. При этом известно, что :

Продолжение таблицы 6.1

Вариант	Условие задачи
	— ни один шарик не лежит в коробочке того же цвета, что и он сам; — в красной коробочке нет синих шариков; — в коробочке нейтрального цвета лежит один красный и один зеленый шарик; нейтральный цвет — это либо белый, либо черный; — в черной коробочке лежат шарики зеленых и синих цветов; — в одной из коробочек лежат один белый и один синий шарик; — в синей коробочке находится один черный шарик. Требуется написать программу, которая распределяла бы шарики по коробочкам в соответствии с вышеуказанными условиями.
7	Пять обладателей выигрышных лотерейных билетов должны получить по два приза. На десяти карточках написали номера от одного до десяти и пригласили каждого счастливчика вытянуть по две карточки. К сожалению, при записи результатов произошла ошибка. В то время, как один из членов тиражной комиссии называл вслух числа, стоявшие на извлеченных карточках (например, : «Пять и семь»), другой по рассеянности складывал эти числа и записывал лишь их сумму (в рассмотренном нами примере это было число 12). Поэтому результаты в пртоколе записаны так : Петров — 11, Семенов — 4, Иванов — 7, Сидоров — 16, Локтев — 17. Требуется определить, какие призы нужно получить каждому обладателю счастливого билета (номер каждого выигранного им приза), если карточки назад не возвращались. Написать программу, возвращающую результат в виде троек значений : Фамилия, Номер первого приза, Номер второго приза.
8	Дама сдавала в багаж : диван, чемодан, саквояж, картину, корзину, картонку и маленькую собачонку. Диван весил столько же, сколько чемодан и саквояж вместе, и столько же, сколько картина и картонка вместе. Картина, корзина и картонка весили поровну, причем каждая из них — больше, чем собачонка. Когда выгружали багаж, дама заявила, что собака не той породы. При проверке оказалось, что собака перевешивает диван, если к ней на весы добавить саквояж или чемодан. Требуется : написать программу, которая доказала бы справедливость претензии дамы.

Продолжение таблицы 6.1

Вариант	Условие задачи
9	После представления «Ревизора» состоялся следующий диалог (В скобках указаны номера фраз-утверждений). Бобчинский : Это вы, Петр Иванович, первый сказали «Э!» (1). Вы сами так говорили (2). Добчинский : Нет, Петр Иванович, я так не говорил (3). Это вы семгу первый заказали (4). Вы и сказали «Э!» (5). А у меня во рту зуб со свистом (6). Бобчинский : Что я семгу первый заказал, это верно (7). И верно, что у вас зуб со свистом (8). А все-таки это вы первый сказали «Э!» (9). Требуется написать программу, определяющую, кто первым сказал «Э!». Известно? что из девяти произнесенных в этом диалоге фраз-утверждений четное число верных.

6.2 **Краткие теоретические сведения.**

6.2.1 **Недетерминированное программирование.**

Определение 6.2.1. *Недетерминизм — техническое понятие, используемое для сжатого определения абстрактных моделей вычислений.*

Недетерминированная машина, перед которой возникло несколько альтернативных путей решения, осуществляет корректный выбор очередного действия, в частности, с применением механизма последовательного поиска и возвратов.

Следует отметить тот факт, что недетерминизм только моделируется, не присутствуя реально.

Примером недетерминированного программирования может послужить метод «образуй и проверь». В этом методе подпрограмма-генератор строит альтернативные пути решения задачи, а подпрограмма-контроллер осуществляет конкретный выбор из ряда альтернатив путем проверки корректности сделанного генератором предположения.

6.2.2 **Суть метода «Образуй и проверь».**

Определение 6.2.2. *«Образуй и проверь» — общий прием проектирования алгоритмов и программ. Суть его состоит в том, что один процесс или программа генерирует множество предполагаемых решений задачи, а другой процесс или программа проверяет эти предполагаемые решения и пытается найти те из них, которые действительно являются решениями задачи.*

Обычно программы, реализующие метод «образуй и проверь», содержат конъюнкцию двух целей, одна из которых действует как генератор

предполагаемых решений, а вторая проверяет допустимость полученных решений :

find(X): — generate(X), test(X).

Если проверка test(X) завершается отказом, то производится возвращение к цели generate, с помощью которой генерируется следующий элемент. Процесс продолжается итерационно до тех пор, пока при успешной проверке не будет найдено решение с требуемыми свойствами или генератор не исчерпает все альтернативные решения.

Стандартный прием оптимизации программ типа «образуй и проверь» заключается в стремлении погрузить программу проверки как можно более «глубоко» в программу генерации. В пределе проверка и генерация реализуется одним правилом, которое порождает только корректные решения.

Пример 1 — задача об офицерах.

На одном вечере среди гостей оказалось пять офицеров : пехотинец, артиллерист, летчик, связист и сапер. Один из них был капитаном, трое майорами и один — в звании подполковника. Из разговоров удалось выяснить следующее :

1) У Якова такое же звание, как у его друга-сапера.
2) Офицер-связист и Филипп — большие друзья.
3) Офицер-летчик вместе с Борисом и Леонидом недавно побывали в гостях у Филиппа.
4) Незадолго до званого вечера у артиллериста и сапера почти одновременно вышли из строя радиоприемники. Оба в один день обратились к Леониду с просьбой зайти к ним и помочь связисту устранить неисправность.
5) Филипп чуть не стал летчиком, но потом по совету своего друга-сапера избрал другой род войск.
6) Яков по званию старше Леонида, а Борис старше Филиппа.
7) Андрей, пятый офицер, накануне вечера был в гостях у Леонида.

Требуется определить :

— звание каждого офицера;
— род войск в котором каждый служит.

Решение.

Условие задачи позволяет определить в виде фактов следующие отношения :

— «принадлежит рассматриваемому множеству офицеров» :

```
oficer(«Яков»).
oficer(«Филипп»).
oficer(«Леонид»).
oficer(«Борис»).
```

oficer(«Андрей»).

— Рода войск, в которых служат рассматриваемые офицеры :

can_be(«Пехотинец»).
can_be(«Артиллерист»).
can_be(«Летчик»).
can_be(«Связист»).
can_be(«Сапер»).

— «Имеет одинаковое звание с кем-либо из рассматриваемых офицеров» :

same(«Яков»).

— «Старше по званию» :

senior(«Яков», «Леонид»).
senior(«Борис», «Филипп»).

— Рода войск, в которых не служит конкретный офицер :

not_was(«Филипп», «Летчик»).
not_was(«Филипп», «Связист»).
not_was(«Филипп», «Сапер»).
not_was(«Яков», «Сапер»").
not_was(«Борис», «Летчик»).
not_was(«Леонид», «Летчик»).
not_was(«Леонид», «Артиллерист»).
not_was(«Леонид», «Сапер»).
not_was(«Леонид», «Связист»).

Порождение и контроль допустимости решений.

Род войск Y, в которых служит офицер X, находится с помощью предиката can_be и затем проверяется, не относится ли данный род войск к тем родам, в которых офицер X не служит :

rod_voisk(X,Y):— oficer(X), can_be(Y), not(not_was(X,Y)).

При определении рода войск, в котором служит каждый из офицеров, род войск для каждого офицера определяется с помощью предиката rod_voisk, а допустимость полученного решения для каждого офицера определяется с помощью предиката condition следующим образом:

```
who_is_who(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5) :—
        rod_voisk(X1,Y1),
        rod_voisk(X2,Y2),
        rod_voisk(X3,Y3),
        rod_voisk(X4,Y4),
        rod_voisk(X5,Y5),
        condition(X1,X2,X3,X4,X5),
        condition(Y1,Y2,Y3,Y4,Y5), !.
```

Определение воинского звания каждого офицера может послужить примером симбиоза проверки и генерации : с помощью предиката oficer

генерируются имя офицера для заданного в заголовке правила zvanie звания, последующая конъюнкция целевых утверждений проверяет допустимость решения.

Листинг программы для решения задачи об офицерах.

```
/* Задача об офицерах */
predicates
        oficer(string)
        zvanie(string,string)
        senior(string,string)
        same(string)
        rod_voisk(string,string)
        can_be(string)
        not_was(string,string)
        condition(string,string,string,string,string)
        who_is_who(string,string,string,string,string,
                        string,string,string,string,string)
clauses
        oficer(«Яков»).
        oficer(«Филипп»).
        oficer(«Леонид»).
        oficer(«Борис»).
        oficer(«Андрей»).

        same(«Яков»).

        senior(«Яков», «Леонид»).
        senior(«Борис», «Филипп»).

        can_be(«Пехотинец»).
        can_be(«Артиллерист»).
        can_be(«Летчик»).
        can_be(«Связист»).
        can_be(«Сапер»).

        not_was(«Филипп», «Летчик»).
        not_was(«Филипп», «Связист»).
        not_was(«Филипп», «Сапер»).
        not_was(«Яков», «Сапер»).
        not_was(«Борис», «Летчик»).
        not_was(«Леонид», «Летчик»).
        not_was(«Леонид», «Артиллерист»).
        not_was(«Леонид», «Сапер»).
        not_was(«Леонид», «Связист»).

        rod_voisk(X,Y):-oficer(X), can_be(Y), not(not_was(X,Y)).

        zvanie(X, «Майор»):-oficer(X), same(X).
```

```
zvanie(X, «Капитан»): — oficer(X), not(same(X)),
            oficer(Y), same(Y), senior(Y,X).
zvanie(X, «Подполковник»): — oficer(X), not(same(X)),
            oficer(X1), same(X1),
            oficer(X2), senior(X1,X2),
            oficer(X3), senior(X,X3).
zvanie(X, «Майор»): — oficer(X), not(same(X)),
            not(zvanie(X, «Капитан»)),
            not(zvanie(X, «Подполковник»)).
condition(X1,X2,X3,X4,X5): —
            X1<>X2, X1<>X3, X1<>X4, X1<>X5,
            X2<>X3, X2<>X4, X2<>X5,
            X3<>X4, X3<>X5, X4<>X5.
who_is_who(X1,Y1,X2,Y2,X3,Y3,X4,Y4,X5,Y5): —
            rod_voisk(X1,Y1),
            rod_voisk(X2,Y2),
            rod_voisk(X3,Y3),
            rod_voisk(X4,Y4),
            rod_voisk(X5,Y5),
            condition(X1,X2,X3,X4,X5),
            condition(Y1,Y2,Y3,Y4,Y5), !.
```

Пример 2 — задача об N ферзях.

Суть задачи : требуется разместить N ферзей на шахматной доске размера N×N так, чтобы на каждой горизонтальной, вертикальной или диагональной линии было не больше одной фигуры. В первоначальной формулировке этой задачи речь шла о размещении 8 ферзей на шахматной доске таким образом, чтобы они, согласно правилам игры в шахматы, не угрожали друг другу.

Для N=2 и N=3 решения не существует, для N=4 решение единственное (без учета симметричного ему решения).

Для реальной шахматной доски 8×8 существует 88 (а с учетом симметричных — 92) решений этой задачи.

В приведенной далее программе отношение queen(N, Qs) истинно, если Qs — решение задачи об N ферзях. Решение представляется некоторой перестановкой списка от 1 до N. Порядковый номер элемента этого списка определяет номер вертикали, а сам элемент — номер горизонтали, на пересечении которых стоит ферзь.

Генерация и проверка допустимости полученных решений.

В состав генератора возможных решений в предлагаемом здесь решении входят предикаты range и select :

```
/* Нахождение списка целых чисел между M и N включительно */
range(N, N, [N]).
range(M, N, [M|Ns]): — M<N, M1=M+1, range(M1,N,Ns).
```

```
/* Выделение элемента из списка */
    select(X, [X|Xs], Xs).
    select(X, [Y|Ys], [Y|Zs]):— select(X, Ys, Zs).
```

Проверка допустимости полученных решений реализуется отрицанием предиката attack. В определении предиката attack использована инкапсуляция взаимосвязи диагоналей.

Два ферзя находятся на одной и той же диагонали на расстоянии N вертикалей друг от друга, если номер горизонтали одного ферзя на N больше или на N меньше, чем номер горизонтали другого ферзя :

```
/* Проверка наличия атаки некоторого ферзя */
    attack(X, Xs):— attack(X, 1, Xs).
    attack(X, N, [Y|Ys]):— X=Y+N; X=Y-N.
    attack(X, N, [Y|Ys]):— N1=N+1, attack(X, N1, Ys).
```

Смысл предиката attack(X, Xs) можно выразить словами : «Ферзь X атакует некоторого другого ферзя из списка Xs». Диагонали проверяются итерационно до тех пор, пока не будет достигнут конец доски.

Последовательное размещение ферзей.

Чтобы проверить, в безопасном ли положении находится новый ферзь, необходимо знать позиции ранее размещенных ферзей. Следовательно, искомое решение должно строиться снизу вверх с применением накапливающего параметра. Использование накапливающего параметра приводит к размещению ферзей, начиная с правой границы доски.

Правило queens осуществляет проверку корректности размещения каждого ферзя непосредственно после генерации его потенциальной позиции на доске :

```
queens(N, Qs):— range(1, N, Ns), queens(Ns, [ ], Qs).

queens([ ], Qs, Qs).

queens(UnplacedQs, SafeQs, Qs):—
   select(Q, UnplacedQs, UnplacedQs1),
   not(attack(Q, SafeQs)),
   queens(UnplacedQs1, [Q|SafeQs], Qs).
```

Листинг программы для решения задачи об N ферзях (реализация на Турбо-Прологе).

```
/* Задача об N ферзях */
domains
    ilist=integer*
predicates
    m(integer, integer)
    d(ilist, integer)
```

```
    t(integer)
    r
    range(integer, integer, ilist)
    select(integer, ilist, ilist)
    attack(integer, ilist)
    attack(integer, integer, ilist)
    queens(integer, ilist)
    queens(ilist, ilist, ilist)
    run_fritz(integer)
    run(integer)
clauses
/* Прорисовка шахматной доски */
    m(X, N):- write(« »), write(«■»), X1=X+1,
              X1<>78, !, m(X1,N).
    m(X, N):- cursor(1,0), write(), !.
    t(0):- !.
    t(X):- X1=X-1, t(X1), cursor(X,0), write().
    r:- attribute(7),
        gotowindow(1), cursor(3,8),
        gotowindow(2).
/* Прорисовка ферзей */
    d([ ], P).
    d([X|T], P):- S=P*2, Q=X-1, cursor(Q,S),
                  attribute(6), writef("@@"),
                  Z=P+1, d(T,Z).
/* Решение задачи об N ферзях */
/* Нахождение списка целых чисел
   между M и N включительно */
    range(N, N, [N]).
    range(M, N, [M|Ns]):- M<N, M1=M+1, range(M1,N,Ns).
/* Выделение элемента из списка */
    select(X, [X|Xs], Xs).
    select(X, [Y|Ys], [Y|Zs]):- select(X, Ys, Zs).
/* Проверка наличия атаки некоторого ферзя */
    attack(X, Xs):- attack(X, 1, Xs).
    attack(X, N, [Y|Ys]):- X=Y+N; X=Y-N.
    attack(X, N, [Y|Ys]):- N1=N+1, attack(X, N1, Ys).
/* Расстановка ферзей на доске */
    queens(N, Qs):- range(1, N, Ns), queens(Ns, [ ], Qs).
    queens([ ], Qs, Qs).
    queens(UnplacedQs, SafeQs, Qs):-
```

```
                select(Q, UnplacedQs, UnplacedQs1),
                not(attack(Q, SafeQs)),
                queens(UnplacedQs1, [Q|SafeQs], Qs).
/* Запуск программы */
    run_fritz(N):—
                gotowindow(1), gotowindow(2), queens(N, Qs),
                m(0,N), t(13), d(Qs,1), N3=N+5, N4=N+N+7,
                makewindow(3,6,0,«»,N3,5,8,30),
                makewindow(4,6,0,«»,5,N4,19,20),
                r, readchar(_), clearwindow,
                gotowindow(4), removewindow,
                gotowindow(3), removewindow, fail.
    run(N):—makewindow(1,6,2,«Задача об N ферзях»,0,0,25,60),
                makewindow(2,6,0,«»,5,5,17,26),
                run_fritz(N).
```

6.2.3 **Решение логических головоломок.**

Решение логических головоломок опишем на примере уже рассмотренной нами задаче об офицерах. Подобные логические головоломки, наряду с моделированием естественных рассуждений средствами логики предикатов 1-го порядка, могут быть решены посредством конкретизации подходящей структуры данных. Проанализируем первый ключ к разгадке : «У Якова такое же звание, как у его друга-сапера». Очевидно, речь идет о двух разных людях. Одного зовут Яков, другой является сапером, но в то же время оба имеют одно и то же воинское звание. В то же время о воинской специальности Якова ничего не известно, хотя из приведенной формулировки понятно, что он — не сапер, поскольку воинские специальности у всех различны. Предположим, что список :

[oficer(«Яков», Z1, R1), oficer(«Филипп», Z2, R2),
oficer(«Борис», Z3, R3), oficer(«Леонид», Z4, R4),
oficer(«Андрей», Z5, R5)]

есть структура данных, подлежащая конкретизации. Тогда наш ключ может быть выражен следующей конъюнкцией целей :

```
can_carry_degree(«Яков», Z1),
can_be(«Яков», R1),
R1<>«Сапер»,
find_eq_zv([oficer(«Филипп», Z2, R2), oficer(«Борис», Z3, R3),
            oficer(«Леонид», Z4, R4), oficer(«Андрей», Z5, R5)], Z1).
```

Посредством применения правила find_eq_zv здесь идет поиск сапера с требуемым воинским званием.

Тот факт, что «офицер-связист и Филипп — большие друзья», говорит о том, что Филипп — не связист. Кроме того, из утверждений :

«офицер-летчик вместе с Борисом и Леонидом недавно побывали в гостях у Филиппа» и

«Филипп чуть не стал летчиком, но потом по совету своего друга-сапера избрал другой род войск»

следует то, что :

1) Филипп, Борис и Леонид — не летчики;
2) Филипп — не сапер.

Относительно воинской специальности Леонида на основании утверждения : «незадолго да званого вечера у артиллериста и сапера почти одновременно вышли из строя радиоприемники и оба в один день обратились к Леониду с просьбой зайти к ним и помочь связисту устранить неисправность» можно заключить, что Леонид не является ни артиллеристом, ни сапером, ни связистом. Сказанное о Филиппе, Борисе и Леониде с учетом утверждения «Яков по званию старше Леонида, а Борис старше Филиппа» выражается приведенной ниже конъюнкцией целей :

```
can_carry_degree(«Яков», Z1),
can_carry_degree(«Филипп», Z2),
can_be(«Филипп», R2),
R2<>«Летчик»,
R2<>«Сапер»,
R2<>«Связист»,
can_carry_degree(«Борис», Z3),
can_be(«Борис», R3),
R3<>«Летчик»,
can_carry_degree(«Леонид», Z4),
can_be(«Леонид», R4),
R4<>«Летчик»,
R4<>«Артиллерист»,
R4<>«Связист»,
R4<>«Сапер»,
senior(Z1,Z4),
senior(Z3,Z2).
```

Здесь с помощью предиката senior формально определяется отношение «старший по званию».

Таким образом, решение задачи об офицерах представляется как последовательное решение ключей с конкретизацией переменных в структурах-элементах списка :

```
[ oficer(«Яков», Z1, R1), oficer(«Филипп», Z2, R2),
oficer(«Борис», Z3, R3), oficer(«Леонид», Z4, R4),
oficer(«Андрей», Z5, R5) ]
```

Программа 2 для решения задачи об офицерах : генератор решений.

```
/* Задача об офицерах. Вариант 2. */
domains
```

```
    slist=string*
    oficer=oficer(string,string,string)
    oficers=oficer*
predicates
    member(string, slist)
    is_a_set(slist)
    counter(string, integer, slist)
    name(string)
    zvanie(string)
    rod_voisk(string)
    can_be(string, string)
    can_carry_degree(string, string)
    senior(string, string)
    find_eq_zv(oficers, string)
    who_is_who(oficers)
    run
clauses
/* Определение принадлежности списку */
    member(H, [H|_]):-!.
    member(Elem, [_ |T]):-
       member(Elem, T).
/* Является ли список множеством ? */
    is_a_set([ ]).
    is_a_set([Head|Tail]):-not(member(Head, Tail)),
                    is_a_set(Tail).
/* Подсчет числа офицеров искомого звания */
    counter(_,0, [ ]).
    counter(Zvanie, N, [Zvanie | Set]):-!,
                    counter(Zvanie, N1, Set),
                    N=N1+1.
    counter(Zvanie, N, [_ | Set ]):-
                    counter(Zvanie, N, Set).
/* Исходные данные для генератора решений */
    name(«Яков»).
    name(«Филипп»).
    name(«Леонид»).
    name(«Борис»).
    name(«Андрей»).
    zvanie(«Майор»).
    zvanie(«Капитан»).
    zvanie(«Подполковник»).
    rod_voisk(«Пехотинец»).
```

```
rod_voisk(«Артиллерист»).
rod_voisk(«Летчик»).
rod_voisk(«Связист»).
rod_voisk(«Сапер»).

can_be(X, Y):—
   name(X),
   rod_voisk(Y).

can_carry_degree(X, Y):—
   name(X),
   zvanie(Y).

/* Определение старшего по званию */

senior(«Майор», «Капитан»).
senior(«Подполковник», «Майор»).
```

Программа 2 для решения задачи об офицерах : контроль допустимости решений.

```
/* Определение наличия среди офицеров сапера
   с заданным воинским званием */

find_eq_zv([ oficer(_, Z, «Сапер»)|_], Z):—!.

find_eq_zv([_|Tail], Z):—
                  find_eq_zv(Tail, Z).

/* Решатель задачи об офицерах */

who_is_who([oficer(«Яков»,Z1,R1), oficer(«Филипп»,Z2,R2),
                  oficer(«Борис»,Z3,R3), oficer(«Леонид»,Z4,R4),
                  oficer(«Андрей»,Z5,R5)]):—

   can_carry_degree(«Яков»,Z1),
   can_be(«Яков»,R1),
   R1<>«Сапер»,
   can_carry_degree(«Филипп»,Z2),
   can_be(«Филипп»,R2),
   R2<>«Летчик»,
   R2<>«Сапер»,
   R2<>«Связист»,
   can_carry_degree(«Борис»,Z3),
   can_be(«Борис»,R3),
   R3<>«Летчик»,
   can_carry_degree(«Леонид»,Z4),
   can_be(«Леонид»,R4),
   R4<>«Летчик»,
   R4<>«Артиллерист»,
   R4<>«Связист»,
   R4<>«Сапер»,
   can_carry_degree(«Андрей»,Z5),
```

```
can_be(«Андрей»,R5),
is_a_set([R1,R2,R3,R4,R5]),
counter(«Майор»,3,[Z1,Z2,Z3,Z4,Z5]),
counter(«Подполковник»,1,[Z1,Z2,Z3,Z4,Z5]),
counter(«Капитан»,1,[Z1,Z2,Z3,Z4,Z5]),
find_eq_zv([oficer(«Филипп»,Z2,R2),
                oficer(«Борис»,Z3,R3), oficer(«Леонид»,Z4,R4),
                oficer(«Андрей»,Z5,R5)],Z1),
senior(Z1,Z4),
senior(Z3,Z2).
```

6.3 **Содержание отчета по работе.**

Отчет по работе должен содержать :

1) Формулировку цели и задач проводимых исследований;
2) Анализ задания, обзор методов решения, выбор метода решения с обоснованием, описание процесса разработки программ, полученные результаты и их анализ (обоснование)
3) Тексты программ с комментариями и обоснованиями.
4) Выводы по проведенным машинным экспериментам.

ЛАБОРАТОРНАЯ РАБОТА №7

ЭКСПЕРТНЫЕ СИСТЕМЫ НА ПРОЛОГЕ

Цель работы.

Целью работы является изучение принципов построения и организации экспертных систем, базирующихся на логике и правилах.

7.1 Задание на лабораторную работу.

На материале лекционного курса и рассмотренного ниже примера экспертной системы изучить методики реализации обоих типов экспертных систем средствами языка Пролог. Реализовать экспертную систему по заданной предметной области в соответствии с номером задания в Таблице 7.1. При этом количество описываемых объектов должно быть не менее 12, а характеризующих их атрибутов — не менее 8. Рассмотреть реализацию экспертной системы как базирующуюся на логике и как базирующуюся на правилах.

Таблица 7.1. Предметная область для экспертной системы

Вариант	Предметная область
1, 5	Микропроцессоры
2, 6	Мобильные устройства
3, 10	Операционные системы
4, 11	Языки программирования
7, 12, 19	Компьютерные игры
8, 15, 20	Компьютерные вирусы
9, 16	Компьютерные сети
13, 17	Алгоритмы сортировки
14, 18	Алгоритмы поиска

7.2 Краткие теоретические сведения.

7.2.1 Назначение и общая структура экспертных систем.

Под Экспертной Системой (ЭС) понимается программа (комплекс программ), моделирующая в некоторой степени работу человека-эксперта в конкретной предметной области. Причем эта область строго ограничена. Основное назначение ЭС — проведение консультаций в той предметной области, для которой спроектирована данная ЭС.

В составе ЭС выделяется три компонента (рис.7.1) :

1) База Знаний (БЗ). БЗ — центральная часть Экспертной Системы. Она содержит совокупность фактов и знаний (правил) для вывода других знаний. Содержащаяся в БЗ информация используется используется ЭС

для вывода экспертного заключения во время консультации. Обычно БЗ располагают отдельно от программы, на диске или другом носителе.

2) Механизм Вывода (МВ). МВ содержит описания способов применения содержащихся в БЗ знаний. Во время консультации МВ запускает ЭС в работу, выполняет правила, определяет приемлемость найденного решения и передает результаты в СПИ.
3) Система Пользовательского Интерфейса (СПИ). СПИ есть та часть ЭС, которая взаимодействует с пользователем. В функции СПИ входит : прием информации от пользователя, передача результатов пользователю в наиболее удобной для него форме, объяснение полученных ЭС результатов (выдача справочной информации по выводу результатов).

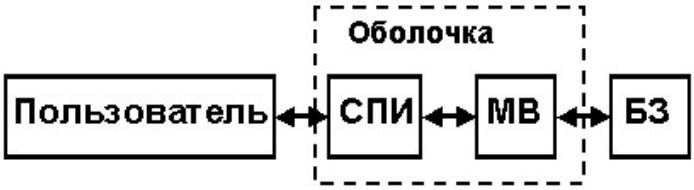

Рис. 7.1. Общая структура ЭС

В зависимости от способов классификации и размещения информации Базы Знаний различают : продукционную, сетевую и фреймовую модели представления знаний.

Сетевая модель основана на представление знаний в виде сети, вершины которой соответствуют понятиям, а дуги — отношениям между ними.

В основе фреймовой модели лежит логическая группировка атрибутов объекта, при этом для хранения и обработки логические группы описываются во фреймах.

Продукционная модель основывается на правилах вида «если–то» и позволяет помещать фрагменты фактического знания в правила языка Пролог. Именно так строятся ЭС, базирующиеся на правилах. При реализации ЭС, базирующейся на логике, БЗ представляется совокупностью утверждений в виде фактов. Вывод экспертного заключения при этом строится на основе стандартных средств работы со списками.

7.2.2 **Вывод экспертного заключения.**

Под выводом в ЭС понимается доказательство того, что из множества предположений следует некоторое заключение. Принятая логика получения заключения специфицируется правилами вывода. Вывод осуществляется посредством поиска и сопоставления по образцу.

В ЭС, базирующейся на правилах, запросы пользователя трансформируются в форму, сопоставимую с формой правил БЗ. Механизм вывода инициализирует процесс сопоставления, начиная с «верхнего» правила. Обращение к правилу называется «вызовом». Вызов соответствующих правил в процессе сопоставления продолжается до тех пор, пока не произошло сопоставление или не исчерпана вся БЗ, а значение не найдено. Если МВ обнаруживает, что можно вызвать более одного правила,

то запускается процесс разрешения конфликта. При разрешении конфликта приоритет отдается обычно тем правилам, которые более конкретны, либо правилам, которые учитывают больше текущих данных.

В ЭС, базирующейся на логике, трансформированные запросы являются значениями, которые сопоставляются со списками значений, находящихся в утверждениях БЗ.

Внутренние программы унификации Турбо-Пролога и Visual Prolog'a позволяют решать задачи поиска и сопоставления по образцу без написания дополнительных правил. Как в системе, базирующейся на правилах, так и в системе, базирующейся на логике, пользователь получает ответы на свои запросы в соответствии с заложенной в ЭС логикой. Для программной реализации механизма вывода экспертного заключения достаточно только написать необходимые спецификации.

7.2.3 **ЭС, базирующаяся на правилах, и ее реализация.**

ЭС, базирующаяся на правилах, позволяет проектировщику строить правила, которые естественным образом объединяют в группы связанные фрагменты знаний. При этом взаимная независимость продукционных правил делает базу правил семантически модульной и способной к развитию.

Реализованная на Турбо-Прологе (или Visual Prolog'e) ЭС на правилах содержит множество правил, которые вызываются посредством входных данных в момент сопоставления. Наряду с множеством правил МВ ЭС имеет в своем составе интерпретатор, который выбирает и активизирует различные модули системы. Работа этого интерпретатора описывается последовательностью трех шагов :

1) Интерпретатор сопоставляет образец правила с элементами данных в БЗ.
2) Если можно вызвать более одного правила, то для выбора правила используется механизм разрешения конфликта.
3) Выбранное правило применяется для поиска ответа на поставленный вопрос.

Этот трехшаговый процесс является циклическим и называется циклом распознавание-действие. Утвердительный ответ ЭС является результатом выполнения одного из продукционных правил, выбор правила производится в соответствии с входными данными.

Написание на Турбо-Прологе (или Visual Prolog'e) ЭС, базирующейся на правилах, начинается с декларации БД. БД хранит ответы пользователя на вопросы СПИ. Эти данные являются утвердительными или отрицательными ответами. Далее строятся продукционные правила, описывающие фрагменты фактического знания. Пример (для ЭС идентификации и выбора породы собак) :

```
dog_is(«английский бульдог»):—
   it_is(«короткошерстная»),
   positive(«имеет», «рост меньше 22 дюймов»),
   positive(«имеет», «свисающий хвост»),
   positive(«имеет», «хороший характер»), !.
/* Аналогично описываются знания о других породах собак */
```

```
...
dog_is(«коккер-спаниэль»): —
   it_is(«длинношерстная»),
   positive(«имеет», «рост меньше 22 дюймов»),
   positive(«имеет», «свисающий хвост»),
   positive(«имеет», «длинные уши»),
   positive(«имеет», «хороший характер»), !.
```

Причем с целью ограничения пространства поиска описывающие связанные фрагменты знаний правила могут бать сгруппированы посредством введения в базу вспомогательных правил для идентификации подкатегорий. К примеру, в ЭС выбора породы собаки это будет правило it_is, которое идентифицирует породу собаки по признаку принадлежности к группе длинношерстных или короткошерстных :

```
it_is(«короткошерстная»): —
   positive(«собака имеет», «короткую шерсть»), !.

it_is(«длинношерстная»): —
   positive(«собака имеет», «длинную шерсть»), !.
```

В рассматриваемом здесь примере основные данные для выбора породы собаки ивляются общеизвестными, поскольку выбраны распространенные породы собак. Набор признаков для классификации пород выбран на основании следующих соображений :

— Все используемые атрибуты являются необходимыми для идентификации породы.
— Ни один из атрибутов не характерен для всех пород одновременно.

Правила МВ сопоставляют данные пользователя с данными в продукционных правилах (правила positive и negative в рассматриваемой ЭС), а также сохраняют «трассу» утвердительных и неутвердительных (отрицательных) ответов (правило remember для добавления в БД утверждений с ответами 1 (да) и 2 (нет)), которые используются при сопоставлении с образцом :

```
/* Механизм вывода экспертного заключения.
   xpositive(X, Y) и xnegative(X, Y) — предикаты динамической БД, хранят,
   соответственно, утвердительные и неутвердительные ответы
   пользователя на вопросы, которые СПИ задает на основе значений
   аргументов предиката positive в теле очередного варианта
   правила dog_is */
  positive(X, Y): —
     xpositive(X,Y), !.
  positive(X, Y): —
     not(negative(X,Y)), !, ask(X,Y).
  negative(X, Y): —
     xnegative(X,Y), !.

/* Консультация. Предикат dlg_Ask создает стандартное диалоговое окно
```

```
получения ответа пользователя на вопрос Да/Нет,
смотри Лабораторную Работу № 4 */
  ask(X, Y):—
    concat(«Вопрос : », X, Temp),
    concat(Temp, « », Temp1),
    concat(Temp1, Y, Temp2),
    concat(Temp2, «?», Quest),
    Reply1=dlg_Ask(«Консультация», Quest, [«Да», «Нет»]),
    Reply=Reply1+1,
    remember(X, Y, Reply).
/* Занесение ответов пользователя в динамическую БД */
  remember(X, Y, 1):—!,
    assertz(xpositive(X, Y)).
  remember(X, Y, 2):—!,
    assertz(xnegative(X, Y)), fail.
```

7.2.4 **ЭС, базирующаяся на логике, и ее реализация.**

Здесь БЗ состоит из утверждений в виде предложений логики предикатов. При этом часть утверждений описывают объекты, другая часть — условия или атрибуты, которые характеризуют различные объекты. Количество признаков определяет степень точности классификации. Интерпретатор внутри системы выполняет свои функции на основе следующей схемы :

1) Система содержит в БЗ предложения, которые управляют поиском и сопоставлением. Интерпретатор сопоставляет эти предложения с элементами данных в БД.
2) Если существует возможность вызова более одного правила, то для разрешения конфликта система использует возможности механизма внутренней унификации Пролога.
3) Система получает результаты унификационного процесса автоматически, поэтому они направляются на нужное (логическое) устройство вывода информации.

Так же, как и в ЭС, базирующейся на правилах, данный циклический процесс является процессом распознавание-действие.

Основное отличие структуры ЭС, базирующейся на логике, состоит в описании объектов и атрибутов в виде фактов :

```
/* Условия-характеристики различных пород.*/
  cond(1, «короткошерстная порода»).
  cond(2, «длинношерстная порода»).
  cond(3, «рост меньше 22 дюймов»).
  cond(4, «рост больше 30 дюймов»).
  cond(5, «свисающий хвост»).
  cond(6, «длинные уши»).
  cond(7, «хороший характер»).
```

```
    cond(8, «вес более 100 фунтов»).
/* Данные о типах породы */
    topic(«короткошерстная»).
    topic(«длинношерстная»).
/* Данные о конкретных породах */
    rule(1, «собака», «короткошерстная», [1]).
    rule(2, «собака», «длинношерстная», [2]).
    rule(3, «короткошерстная», «английский бульдог», [3,5,7]).
    rule(4, «короткошерстная», «гончая», [3,6,7]).
    rule(5, «короткошерстная», «датский дог», [5,6,7,8]).
    rule(6, «короткошерстная», «американский фокстерьер», [4,6,7]).
    rule(7, «длинношерстная», «коккер-спаниэль», [3,5,6,7]).
    rule(8, «длинношерстная», «ирландский сеттер», [4,6]).
    rule(9, «длинношерстная», «колли», [4,5,7]).
    rule(10, «длинношерстная», «сенбернар», [5,7,8]).
```

Третий аргумент предиката rule — список целых чисел-номеров условий из предложений типа cond. Каждое предложение типа cond задает свое условие-признак из используемых здесь для классификации пород собак. В ЭС, базирующейся на логике, интерпретатор использует эти номера условий, чтобы делать соответствующий выбор.

МВ содержит правила обработки списков атрибутов в описаниях объектов. Посредством применения правила go МВ просматривает утверждения БЗ rule и cond для выяснения с помощью правила check существования или отсутствия подходящих значений данных :

```
/* Начальное правило механизма вывода */
    go(_, Mygoal):—
      not(rule(_, Mygoal, _, _)), !,
      concat(«Рекомендуемая порода : », Mygoal, Temp),
      concat(Temp, «.», Result),
      dlg_Note(«Экспертное заключение : », Result).
    go(History, Mygoal):—
      rule(Rule_number, Mygoal, Type_of_breed, Conditions),
      check(Rule_number, History, Conditions),
      go([Rule_number|History], Type_of_breed).
/* Сопоставление входных данных пользователя
   со списками атрибутов отдельных пород собак */
    check(Rule_number, History, [Breed_cond|Rest_breed_cond_list]):—
      yes(Breed_cond), !,
      check(Rule_number, History, Rest_breed_cond_list).
    check(_, _, [Breed_cond|_]):—
      no(Breed_cond), !, fail.
    check(Rule_number, History, [Breed_cond|Rest_breed_cond_list]):—
      cond(Breed_cond, Text),
```

```
    ask_question(Breed_cond, Text),
    check(Rule_number, History, Rest_breed_cond_list).

check(_, _, [ ]).

do_answer(Cond_number, 1):— !,
    assertz(yes(Cond_number)).

do_answer(Cond_number, 2):— !,
    assertz(no(Cond_number)), fail.

/* Запрос и получение ответов yes и no от пользователя */

ask_question(Breed_cond, Text):—
    concat(«Вопрос : », Text, Temp),
    concat(Temp, « », Temp1),
    concat(Temp1, «?», Quest),
    Response1=dlg_Ask(«Консультация», Quest, [«Да»,«Нет»]),
    Response=Response1+1,
    do_answer(Breed_cond, Response).
```

Правило go пытается сопоставить объекты, классифицированные при помощи номеров условий. Если сопоставление происходит, то данный модуль программы должен добавить в БЗ сопоставленные значения и продолжить процесс с новыми данными, полученными от пользователя. Если сопоставление не происходит, механизм останавливает текущий процесс и выбирает для сопоставления другую трассу. Поиск и сопоставление продолжается до тех пор, пока не исчерпаны все возможности.

Достоинством ЭС, базирующейся на логике, является возможность хранения фактов Базы Знаний в утверждениях динамической Базы Данных. При этом База Знаний может быть размещена в файле на диске, что позволяет сделать ее не зависимой от программного кода.

7.3 **Содержание отчета по работе.**

Отчет по работе должен содержать :

1) Формулировку цели и задач проводимых исследований;
2) Описание характеристик разработанных Экспертных Систем;
3) Диаграмма потоков данных и структурная диаграмма для вариантов реализации Экспертной Системы как базирующейся на правилах и как базирующейся на логике;
4) Тексты программ с комментариями и обоснованиями;
5) Описание механизмов вывода в рассмативаемых Экспертных Системах;
6) Выводы по проведенным машинным экспериментам.

ЛАБОРАТОРНАЯ РАБОТА №8

ИНТЕРФЕЙС НА ЕСТЕСТВЕННОМ ЯЗЫКЕ В ЭКСПЕРТНЫХ СИСТЕМАХ

Цель работы.

Целью работы является изучение лингвистических возможностей языка Пролог.

8.1 Задание на лабораторную работу.

Дополнить реализованную в Лабораторной работе №7 Экспертную Систему пользовательским интерфейсом на Естественном (русском) Языке. Подсистема понимания Естественного Языка реализуется на основе описанной ниже стратегии с использованием ключевых слов. Для формирования пользователем запроса на русском языке в произвольной форме в состав системы должен быть включен текстовый редактор.

8.2 Краткие теоретические сведения.

8.2.1 Проблема понимания компьютером Естественного Языка.

Под Естественным Языком (ЕЯ) в лингвистике понимается любой язык общения между людьми. Под естественностью некоторого языка понимается наличие синонимии и омонимии слов и словосочетаний, а также возможность присутствия свободного порядка слов в предложении.

Целью исследований в области обработки естественного языка является построение программных средств, обеспечивающих восприятие и генерацию ЕЯ-текстов. При этом текст рассматривается как подлежащее восприятию дискретное представление ЕЯ-высказывания. ЕЯ-высказывание здесь представляется в том виде, в котором текстовый материал вводится с клавиатуры. Отдельную область исследований составляет звуковое распознавание и синтез звучащей речи как перевод речевого сигнала в текстовое представление и наоборот.

8.2.2 Виды анализа ЕЯ-информации.

Наиболее известными являются следующие подходы к решению задач ЕЯ-общения конечного пользователя с ЭВМ :

— Прагматический анализ - наиболее сложный, связан с изучением смысла предложения в связи с внеязыковой действительностью. Анализ ключевых слов — метод анализа ЕЯ-высказываний на предмет наличия ключевых слов, которые становятся значениями аргументов предикатов. При этом компьютер одинаково реагирует на различные варианты входного текста, наличие грамматической правильности предложений не является обязательным, роль играет лишь наличие ключевых слов. Применение - построение ЕЯ-интерфейсов к Базам Данных (БД).

— Грамматический анализ : контекстно-свободный и контекстно-зависимый. Контекстно-Свободный (КС) анализ — ЕЯ-фразы классифицируются в зависимости от их внутренней структуры вне зависимости от контекста в соответствии с грамматическими правилами, задающими порядок следования допустимых языком символов (слов). Здесь следует отметить синтаксический анализ предложений.
— Прагматический анализ — наиболее сложный, связан с изучением смысла предложения с учетом связи с внеязыковой действительностью.

8.2.3 ЕЯ-интерфейс к Структурированным Источникам Данных на основе ключевых слов.

Среди известных на сегодняшний день типов Структурированных Источников Данных (СИД) [2] следует выделить реляционные и объектные СУБД, XML и RDF.

Наиболее распространенными типами СИД на сегодняшний день являются реляционные и объектные Базы Данных (БЛ). XML и RDF относятся к Internet-хранилищам, для которых характерна меньшая степень структуризации, чем для Баз Данных [2].

При разработке ЕЯ-интерфейса к БД на основе ключевых слов с ключевым словом связывается либо характер действия программы (что хочет пользователь), либо значение некоторого атрибута искомого объекта.

Процедура анализа входного текста с целью выделения ключей состоит из трех шагов :

— Ввод команды как строки символов.
— Преобразование введенной строки в список слов.
— Идентификация ключевых слов.

Для преобразования строки в список слов требуется написание правила, которое преобразует строку в список. Для этой цели служит встроенный предикат fronttoken. Однако данный предикат предназначен для работы с предложениями на английском языке и не работает со строками, записанными на русском. Здесь программист должен написать свою процедуру разделения русского предложения на слова.

Рассмотрим пример реализации такой прецедуры средствами Visual Prolog'a.

```
/* Выделение подстроки до первого разделителя */
    fronttoken_cyr(Str, Token, Rest_of_string):—
      symbol_counter(Str, Number),
      frontstr(Number,Str, Token,Rest_of_string).

/* Подсчет символов в строке до конца строки, либо ближайшего пробела,
символа возврата каретки, перевода строки, !,«»,#,$ */

    symbol_counter(«», 0).

/* Разделитель в строке — пример для пробела */

    symbol_counter(Str, 0):—
      frontchar(Str, '\32', _), !.
```

```
/* Аналогично описывается условие окончания рекурсии для других
разделителей */
        ...

    symbol_counter(Str,Number):-
      frontchar(Str, _Char, Rest_of_string),
      symbol_counter(Rest_of_string, Number1),
      Number=Number1+1.

/* Удаление разделителя в начале строки */

    del_front_space(«»,«»).
    del_front_space(Arg, Res):-
      frontchar(Arg, Char, Res1), Char='\32', !,
      del_front_space(Res1, Res).

/* Аналогично описывается условие окончания рекурсии для других
разделителей */
        ...

    del_front_space(Arg, Arg):-frontchar(Arg, Char, _),
      not(member(Char,['\32','\10','\13','\33',
                      '\34','\35','\36','\40','\41',
                      '\44','\45','\46','\59'])).

/* Предикат upper_lower для кирилицы Windows */

    upper_lower_cyr(InString, OutString):-
      str_char_list(InString,Char_List_for_InString),
      upper_lower_cyr_convers(Char_List_for_InString,
                              Char_List_for_OutString),
      pack(Char_List_for_OutString,OutString).

    upper_lower_cyr_convers([],[]).

    upper_lower_cyr_convers([Char|Char_List],[Char1|Char_List1]):-
      char_int(Char,ASCII_code),
      ASCII_code>=192,ASCII_code<=223,!,
      ASCII_code_new=ASCII_code+32,
      char_int(Char1,ASCII_code_new),
      upper_lower_cyr_convers(Char_List,Char_List1).

    upper_lower_cyr_convers([Char|Char_List],[Char1|Char_List1]):-
      char_int(Char,ASCII_code),
      ASCII_code=168,!,
      ASCII_code_new=ASCII_code+16,
      char_int(Char1,ASCII_code_new),
      upper_lower_cyr_convers(Char_List,Char_List1).

    upper_lower_cyr_convers([Char|Char_List],[Char1|Char_List1]):-
      char_int(Char,ASCII_code),
      ASCII_code>=65, ASCII_code<=90, !,
      ASCII_code_new=ASCII_code+32,
      char_int(Char1,ASCII_code_new),
```

```
    upper_lower_cyr_convers(Char_List,Char_List1).
upper_lower_cyr_convers([Char|Char_List],[Char|Char_List1]):-
    upper_lower_cyr_convers(Char_List, Char_List1).
```

/* Преобразование строки в список символов */

```
str_char_list(«»,[]).
str_char_list(Word,[Char|Char_List]):-
    frontchar(Word,Char,WordRest),
    str_char_list(WordRest,Char_List).
```

/* Превращение списка символов в строку */

```
pack([ ],«»).
pack([H|T], Res):-
    str_char(Str_H, H),
    pack(T, Res1),
    concat(Str_H, Res1, Res).
```

/* Модифицированное правило преобразования строки в список слов */

```
convers(«»,[ ]):-!.
convers(Str, [Head1|Tail]):-
    fronttoken_cyr(Str, Head, Str2),
    upper_lower_cyr(Head, Head1),
    del_front_space(Str2, Str1),
    convers(Str1, Tail).
```

Идентификация ключевых слов может быть реализована двумя способами. Первый состоит в задании всех ключевых слов в утверждениях БД. Тогда внутренние унификационные процедуры Пролога смогут сопоставить элементы предложения со значениями, взятыми из БД. Второй способ основан на частоте применения определенных грамматических конструкций : ключевые слова определяются позицией, которую они занимают в предложении. В примере для реализуемой нами Экспертной Системы в качестве ключевых берутся первое (Kname) и последнее (Lname) слово ЕЯ-высказывания. При этом первое слово ассоциируется с выполняемым программой действием, второе — с объектом из БД, над которым производится действие.

Пример для варианта Экспертной Системы, базирующегося на логике :

/* Правило для проверки ключевых слов */

```
do_right_form(Kname, Lname):-
    func_keyword(Kname),
    rule(_,Lname,_,_), !.
do_right_form(Kname,_):-
    frontstr(3, Kname, Word, _),
    upper_lower_cyr(Word, Key),
    Key=«вых»,!,
    exit.
```

```
do_right_form(Kname,_):-
   upper_lower_cyr(Kname, Word),
   Word=«exit», !,
   exit.

do_right_form(Kname,_):-
   upper_lower_cyr(Kname, Word),
   Word=«quit», !,
   exit.

do_right_form(Kname, Lname):-
   concat(«Введенные Вами ключевые слова : », Kname, Temp),
   concat(Temp, « и »,Temp1),
   concat(Temp1, Lname, Temp2),
   concat(Temp2, « не известны системе. », Msg),
   write(Msg), !.
```

```
/* Пример проверки допустимости ключевого слова-приказа */

func_keyword(Word):-
   frontstr(3, Word, Key, _),
   upper_lower_cyr(Key, KeyTr),
   KeyTr=«най», !.
```

8.3 **Создание окна текстового редактора в Visual Prolog.**

В Visual Prolog для создания текстовых редакторов эксперт окон и диалоговых окон в составе среды визуальной разработки имеет особый стиль программного кода, который генерируется экспертом кода для окна редактора, стиль edit_Create. Для настройки и использования редактора существует гибкий API (Application Programming Interface) [1, с. 680].

При настройке созданного редактора рекомендуется в сгенерированном по умолчанию коде окна редактора указать задаваемую предикатом font_Create семейство и размер шрифта для редактируемого текста (в частности, с учетом наличия поддержки кириллицы). Кроме того, в генерируемом по умолчанию коде предиката создания окна редактора отсутствует указание источника редактируемого текста. Редактируемый текст можно сделать одним из аргументов указанного предиката. Пример :

```
win_грамматический_разбор_Create(_Parent, Text):-
ifdef use_editor
   Font = font_Create(ff_System, [ ], 10),
   ReadOnly = b_false, Indent = b_true, InitPos = 1,
   edit_Create(win_грамматический_разбор_WinType,
            win_грамматический_разбор_RCT,
            win_грамматический_разбор_Title,
            win_грамматический_разбор_Menu,
            _Parent, win_грамматический_разбор_Flags, Font,
            ReadOnly,
            Indent, Text, InitPos, win_грамматический_разбор_eh),
```

```
enddef

    true.
```

8.4 **Содержание отчета по работе.**

1) Формулировку цели и задач проводимых исследований;
2) Анализ задания, выбор метода решения с обоснованием, описание процесса разработки программ, полученные результаты и их анализ (обоснование);
3) Тексты программ с комментариями и обоснованиями;
4) Выводы по проведенным машинным экспериментам.

СПИСОК ЛИТЕРАТУРЫ

1 Адаменко А. Н., Кучуков А. М. Логическое программирование и Visual Prolog. — СПб. : БХВ-Петербург, 2003. — 992 с. : ил.

2 Жигалов В. А., Соколова Е. Г. InBASE:ТЕХНОЛОГИЯ ПОСТРОЕНИЯ ЕЯ-ИНТЕРФЕЙСОВ К БАЗАМ ДАННЫХ // Труды международного семинара Диалог'2001 по компьютерной лингвистике и ее приложениям. — 2001. — т.2, с. 123–135.

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

4 Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог : Пер. с англ. — М. : Мир, 1990. — 235 с. : ил.

5 Нильсон Н. Искусственный интеллект. Методы поиска решений : Пер. с англ. — М. : Мир, 1973. — 270 с. : черт.

6 Заочные математические олимпиады. — 2-е изд., перераб. / Н. Б. Васильев, В. Л. Гутенмахер, Ж. М. Раббот, А. Л. Тоом. — М. : Наука, 1987. — 176 с.

7 Гик Е. Я. Шахматы и математика. — М. : Наука, 1983. — 175 с. : ил.

8 Арсак Ж. Программирование игр и головоломок. — М. : Наука, 1990. — 220 с. : ил.

9 Гарднер М. Математические досуги. — М. : Оникс, 1995. — 495 с. : ил.

10 Демидович Б. П. Сборник задач и упражнений по математическому анализу. — М. : Наука, 1977. — 527 с.

11 Мочалов Л. П. Головоломки : Кн. для учащихся. — М. : Просвещение : АО «Учеб. лит.», 1996. — 190 с. : ил.

12 Вирт Н. Алгоритмы + структуры данных = программы : Пер. с англ. — М. : Мир, 1985. — 406 с. : ил.

13 Вирт Н. Алгоритмы и структуры данных : Пер. с англ. — СПб. : Невский диалект, 2001. — 351 с.

14 Ахо А. В., Хопкрофт Д. Э., Ульман Д. Д. Структуры данных и алгоритмы : Пер. с англ. — М. : Издательский дом «Вильямс», 2000. — 382 с.

15 Маплас Дж. Реляционный язык Пролог и его применение : Пер. с англ. — М. : Наука, 1990. — 464 с.

Приложение А

Пример реализации графического пользовательского интерфейса средствами Visual Prolog

Головной модуль программы (файл test.pro).

```
include «test.inc»
include «test.con»
include «hlptopic.con»
/*************************************************************************
      Обработка событий для окна Task Window
*************************************************************************/
predicates
      task_win_eh : EHANDLER
constants
/*BEGIN Task Window,CreateParms,19:14:00–15.12.2007, Code automatically
updated!*/
      task_win_Flags = [wsf_SizeBorder,wsf_TitleBar,wsf_Close,
                        wsf_Maximize,wsf_Minimize,wsf_ClipSiblings]
      task_win_Menu = res_menu(idr_task_menu)
      task_win_Title = «Демо-интерфейс»
      task_win_Help = idh_contents
/* END Task Window, CreateParms */
clauses
/* BEGIN Task Window, e_Create */
      task_win_eh(_Win,e_Create(_),0):-!,
/* BEGIN Task Window, InitControls, 19:14:00–15.12.2007,
      Code automatically updated ! */
/* END Task Window, InitControls */
/* BEGIN Task Window, ToolbarCreate, 19:14:00–15.12.2007,
      Code automatically updated ! */
         tb_project_toolbar_Create(_Win),
         tb_help_line_Create(_Win),
/* END Task Window, ToolbarCreate */
ifdef use_message
         msg_Create(100),
enddef
         !.
/* END Task Window, e_Create */
/* MARK Task Window, new events */
/* BEGIN Task Window, id_демокнопки */
      task_win_eh(_Win,e_Menu(id_демокнопки,_ShiftCtlAlt),0):-!,
```

```
                    win_кнопки_для_реализации_игры_Create(_Win),!.
/* END Task Window, id_демокнопки */
/* BEGIN Task Window, id_edit_cut */
    task_win_eh(_Win,e_Menu(id_edit_cut,_ShiftCtlAlt),0):—!,
      Msg=«Вводимая строка : »,
      InitStr=«Строка по умолчанию»,
      Title1=«Ввод строки»,
      NewSTRING=dlg_GetStr(Title1,Msg,InitStr),
      SLIST=[«Пункт 1»,«Пункт 2»,«Пункт 3»],
      PreSel=1,
      dlg_ListSelect(NewSTRING,SLIST,PreSel,StrSel,_Index),
      dlg_Note(NewSTRING,StrSel),!.
/* END Task Window, id_edit_cut */
/* BEGIN Task Window, id_help_contents */
    task_win_eh(_Win,e_Menu(id_help_contents,_ShiftCtlAlt),0):—!,
      vpi_ShowHelp(«test.hlp»),
      !.
/* END Task Window, id_help_contents */
/* BEGIN Task Window, id_file_exit */
    task_win_eh(Win,e_Menu(id_file_exit,_ShiftCtlAlt),0):—!,
      win_Destroy(Win),
      !.
/* END Task Window, id_file_exit */
/* BEGIN Task Window, e_Size */
    task_win_eh(_Win,e_Size(_Width,_Height),0):—!,
ifdef use_tbar
      toolbar_Resize(_Win),
enddef
ifdef use_message
      msg_Resize(_Win),
enddef
      !.
/* END Task Window, e_Size */
/* END_WIN Task Window */
/*****************************************************************
    Invoking on-line Help
*****************************************************************/
    project_ShowHelpContext(HelpTopic):—
      vpi_ShowHelpContext(«test.hlp»,HelpTopic).
/*****************************************************************
    Main Goal
*****************************************************************/
goal
```

```
ifdef use_mdi
        vpi_SetAttrVal(attr_win_mdi,b_true),
enddef
ifdef ws_win
      ifdef use_3dctrl
        vpi_SetAttrVal(attr_win_3dcontrols,b_true),
      enddef
enddef
        vpi_Init(task_win_Flags,task_win_eh,task_win_Menu,
                 «test»,task_win_Title).
/* BEGIN_TLB Help line, 13:03:58–15.12.2007, Code automatically updated! */
/*****************************************************************
      Creation of toolbar : Help line
*****************************************************************/
clauses
      tb_help_line_Create(_Parent) :-
ifdef use_tbar
        toolbar_create(tb_bottom,0xC0C0C0,_Parent,
             [tb_text(idt_help_line,tb_context,452,0,4,10,0x0,«»)]),
enddef
        true.
/* END_TLB Help line */
/* BEGIN_DLG About dialog */
/*****************************************************************
      Creation and event handling for dialog : About dialog
*****************************************************************/
constants
/* BEGIN About dialog, CreateParms, 13:04:00–15.12.2007, Code automatically
updated! */
      dlg_about_dialog_ResID = idd_dlg_about
      dlg_about_dialog_DlgType = wd_Modal
      dlg_about_dialog_Help = idh_contents
/* END About dialog, CreateParms */
predicates
      dlg_about_dialog_eh : EHANDLER
clauses
      dlg_about_dialog_Create(Parent) :-
        win_CreateResDialog(Parent,dlg_about_dialog_DlgType,
                            dlg_about_dialog_ResID,dlg_about_dialog_eh,0).
/* BEGIN About dialog, idc_ok _CtlInfo */
      dlg_about_dialog_eh(_Win,e_Control(idc_ok,_CtrlType,
                          _CtrlWin,_CtrlInfo),0) :-!,
        win_Destroy(_Win),!.
/* END About dialog, idc_ok _CtlInfo */
```

```
/* MARK About dialog, new events */
    dlg_about_dialog_eh(_,_,_) :-!,fail.
/* END_DLG About dialog */
/* BEGIN_WIN Test_draw */
/************************************************************
    Creation and event handling for window: Test_draw
************************************************************/
constants
/* BEGIN Test_draw, CreateParms, 13:19:47–15.12.2007,
    Code automatically updated ! */
    win_test_draw_WinType = w_TopLevel
    win_test_draw_Flags = [wsf_SizeBorder,wsf_TitleBar,wsf_Maximize,
                           wsf_Minimize,wsf_Close,wsf_ClipSiblings,
                           wsf_ClipChildren,wsf_VScroll,wsf_HScroll]
    win_test_draw_RCT = rct(100,80,440,240)
    win_test_draw_Menu = no_menu
    win_test_draw_Title = «Test_draw»
    win_test_draw_Help = idh_contents
/* END Test_draw, CreateParms */
predicates
    win_test_draw_eh : EHANDLER
clauses
    win_test_draw_Create(_Parent) :-
      win_Create(win_test_draw_WinType,win_test_draw_RCT,
                 win_test_draw_Title,win_test_draw_Menu,
                 _Parent,win_test_draw_Flags,win_test_draw_eh,0).
/* BEGIN Test_draw, e_Create */
    win_test_draw_eh(_Win,e_Create(_),0) :-!,
      draw_Polygon(_Win,[pnt(2,2),pnt(6,6),pnt(4,3),pnt(2,6)]),
/* BEGIN Test_draw, InitControls, 13:19:47–15.12.2007,
    Code automatically updated ! */
/* END Test_draw, InitControls */
/* BEGIN Test_draw, ToolbarCreate, 13:19:47–15.12.2007,
    Code automatically updated ! */
/* END Test_draw, ToolbarCreate */
      !.
/* END Test_draw, e_Create */
/* MARK Test_draw, new events */
/* BEGIN Test_draw, e_Size */
    win_test_draw_eh(_Win,e_Size(_Width,_Height),0) :-!,
```

```
ifdef use_tbar
        toolbar_Resize(_Win),
enddef
        !.
/* END Test_draw, e_Size */
/* BEGIN Test_draw, e_Menu, Parent window */
      win_test_draw_eh(Win,e_Menu(ID,CAS),0) :- !,
        PARENT = win_GetParent(Win),
        win_SendEvent(PARENT,e_Menu(ID,CAS)),!.
/* END Test_draw, e_Menu, Parent window */
/* END_WIN Test_draw */
/* BEGIN_TLB Project toolbar, 13:28:02–16.12.2007,
      Code automatically updated ! */
/*****************************************************************
      Creation of toolbar: Project toolbar
*****************************************************************/
clauses
      tb_project_toolbar_Create(_Parent) :-
ifdef use_tbar
        toolbar_create(tb_top,0xC0C0C0,_Parent,
            [tb_ctrl(id_file_new,pushb,idb_new_up,idb_new_dn,idb_new_up,
                    «New;New file»,1,1),
            tb_ctrl(id_file_open,pushb,idb_open_up,idb_open_dn,
                    idb_open_up,«Open;Open file»,1,1),
            tb_ctrl(id_file_save,pushb,idb_save_up,idb_save_dn,
                    idb_save_up,«Save;File save»,1,1),
            separator,
            tb_ctrl(id_edit_undo,pushb,idb_undo_up,idb_undo_dn,
                    idb_undo_up,«Undo;Undo»,1,1),
            tb_ctrl(id_edit_redo,pushb,idb_redo_up,idb_redo_dn,
                    idb_redo_up,«Redo;Redo»,1,1),
            separator,
            tb_ctrl(id_edit_cut,pushb,idb_cut_up,idb_cut_dn,
                    idb_cut_up,«Cut;Cut to clipboard»,1,1),
            tb_ctrl(id_edit_copy,pushb,idb_copy_up,idb_copy_dn,
                    idb_copy_up,«Copy;Copy to clipboard»,1,1),
            tb_ctrl(id_edit_paste,pushb,idb_paste_up,idb_paste_dn,
                    idb_paste_up,«Paste;Paste from clipboard»,1,1),
            separator,
            separator,
            tb_ctrl(id_help_contents,pushb,idb_help_up,idb_help_down,
                    idb_help_up,«Help;Help»,1,1)]),
enddef
      true.
/* END_TLB Project toolbar */
```

Обработка событий окна «Кнопки для реализации игры» (файл crosses_zeros_demo.pro).

```
include «test.inc»
include «test.con»
include «hlptopic.con»
/* BEGIN_WIN Кнопки для реализации игры */
/*************************************************************
    Обработка событий для окна Кнопки для реализации игры
*************************************************************/
constants
/* BEGIN Кнопки для реализации игры, CreateParms, 13:16:02–16.12.2007,
    Code automatically updated! */
    win_кнопки_для_реализации_игры_WinType = w_TopLevel
    win_кнопки_для_реализации_игры_Flags = [wsf_TitleBar,
                                            wsf_Border,
                                            wsf_Close]
    win_кнопки_для_реализации_игры_RCT = rct(100,80,449,259)
    win_кнопки_для_реализации_игры_Menu = no_menu
    win_кнопки_для_реализации_игры_Title = «Кнопки
                                           для реализации игры»
    win_кнопки_для_реализации_игры_Help = idh_contents
/* END Кнопки для реализации игры, CreateParms */
predicates
    win_кнопки_для_реализации_игры_eh : EHANDLER
clauses
    win_кнопки_для_реализации_игры_Create(_Parent) :- !,
      win_Create(win_кнопки_для_реализации_игры_WinType,
                 win_кнопки_для_реализации_игры_RCT,
                 win_кнопки_для_реализации_игры_Title,
                 win_кнопки_для_реализации_игры_Menu,
                 _Parent,
                 win_кнопки_для_реализации_игры_Flags,
                 win_кнопки_для_реализации_игры_eh,0),!.
/* BEGIN Кнопки для реализации игры, e_Create*/
    win_кнопки_для_реализации_игры_eh(WIN1,e_Create(_),0):- !,
      win_SetActiveWindow(WIN1),
      win_SetFocus(WIN1),
      win_CreateControl(wc_PushButton,rct(98,50,148,100),«1»,WIN1,
                 [wsf_UpperCase],idc_1),
      win_CreateControl(wc_PushButton,rct(49,50,99,100),«2»,WIN1,
                 [wsf_UpperCase],idc_2),
      win_CreateControl(wc_PushButton,rct(150,50,200,100),«3»,WIN1,
                 [wsf_UpperCase,wsf_Disabled],idc_3),
      win_CreateControl(wc_Text,rct(50,150,200,170),«3 — не изменяется»,
                 WIN1,[wsf_UpperCase],idct_3_не_изменяется),
      win_Update(WIN1),!.
```

```
/* END Кнопки для реализации игры, e_Create */
/* MARK Кнопки для реализации игры, new events */
/* BEGIN Кнопки для реализации игры, idc_2 _CtlInfo */
    win_кнопки_для_реализации_игры_eh(_Win,e_Control(idc_2,
                                      wc_PushButton,_,
                                      _ShiftCtrlAlt),idc_2) :- !,
      write(«Крестик»),
      win_SetBrush(_Win,brush(pat_Solid,color_White)),
      win_SetPen(_Win,pen(5,ps_Solid,color_White)),
      draw_Rect(_Win,rct(155,55,195,95)),
      win_SetPen(_Win,pen(5,ps_Solid,color_Blue)),
      draw_FloodFill(_Win,pnt(175,75),color_White),
      draw_Line(_Win,pnt(160,60),pnt(190,90)),
      draw_Line(_Win,pnt(160,90),pnt(190,60)),!.
/* END Кнопки для реализации игры, idc_2 _CtlInfo */
/* BEGIN Кнопки для реализации игры, idc_1 _CtlInfo */
    win_кнопки_для_реализации_игры_eh(_Win,e_Control(idc_1,
                                      wc_PushButton,_,_ShiftCtrlAlt),idc_1):-
      !,write(«Нолик»),
      win_SetBrush(_Win,brush(pat_Solid,color_White)),
      win_SetPen(_Win,pen(5,ps_Solid,color_White)),
      draw_Rect(_Win,rct(155,55,195,95)),
      win_SetPen(_Win,pen(5,ps_Solid,color_Green)),
      draw_FloodFill(_Win,pnt(175,75),color_White),
      draw_Ellipse(_Win,rct(160,60,190,90)),!.
/* END Кнопки для реализации игры, idc_1 _CtlInfo */
/* BEGIN Кнопки для реализации игры, e_Size */
    win_кнопки_для_реализации_игры_eh(_Win,e_Size(_Width,
                                      _Height),0):- !,
ifdef use_tbar
      toolbar_Resize(_Win),
enddef
      !.
/* END Кнопки для реализации игры, e_Size */
/* BEGIN Кнопки для реализации игры, e_Menu, Parent window */
    win_кнопки_для_реализации_игры_eh(Win,e_Menu(ID,CAS),0):- !,
      PARENT = win_GetParent(Win),
      win_SendEvent(PARENT,e_Menu(ID,CAS)),
      !.
/* END Кнопки для реализации игры, e_Menu, Parent window */
/* END_WIN Кнопки для реализации игры */
```