1. ПРЕДМЕТ ЛОГИЧЕСКОГО ПРОГРАММИРОВАНИЯ

Традиционные языки программирования основываются на структуре компьютера и командах, которые может выполнить компьютер, поэтому человек должен обучаться алгоритмическому мышлению.

Логическое программирование исходит из того, что компьютер должен уметь работать по логическим построениям, присущим человеку. Например, в логическом программировании разрешена конструкция типа *"Определить главную причину снижения студенческой успеваемости"*. Данной конструкции достаточно, чтобы получить ответ.

Логическое программирование – это подход к программированию, при котором программа задаётся совокупностью правил без явного указания последовательности их применения.

Под логическим программированием обычно подразумевают идею использования для создания программного обеспечения логики предикатов, в частности рассуждений, которые показывают путь к достижению желаемого результата.

Предложение естественного языка без труда может быть преобразовано в предложение логики предикатов. Для этого следует сначала устранить все ненужные слова из предложения, после чего преобразовать предложение так, чтобы свойство или отношение стояло первым, а обладающие данным свойством или находящиеся в данном отношении объекты были сгруппированы после него. Тогда объекты становятся параметрами, на которых это свойство или отношение действует. В табл.1 предложения естественного языка преобразованы в синтаксис логики предикатов.

Таблица 1

Преобразование предложений естественного языка

Естественный язык	Логика предикатов
Роза красная.	красная (роза).
Монополия – игра.	игра (монополия).
Ане нравится монополия, если монополия – игра.	нравится (аня, монополия) if игра (монополия).

У языков логического программирования два главных отличия от классических алгоритмических языков:

1. *Это символьное программирование*. Данные в нём суть символы, представляющие только самих себя и не подлежащие интерпретации при выполнении программы.
2. *Алгоритмы* получения определённых результатов непосредственно *не задаются*, но описываются объекты, их свойства и отношения.

Целью курса является изучение и практическое освоение логического программирования для решения научных и прикладных задач. В качестве инструментального средства изучается язык *Пролог*. Рассматриваются теоретические и прикладные аспекты использования *Пролога* для решения практических задач.

Основные задачи курса связаны с приобретением навыков формализации решаемых задач:

- выделение понятий предметной или проблемной областей;
- формирование определений понятий для человека и базы знаний;
- составление характеристик понятий как множества значений;
- построение фраз естественного языка для фиксации знаний;
- автоматическая отладка и кодирование знаний и данных;
- автоматический поиск результатов логического вывода ответа на запрос, сводящийся к решению логического уравнения.

1.1. Предпосылки появления логического программирования

Одной из главных предпосылок развития логического программирования явилось существование большого и очень важного класса неформализованных задач. К неформализованным, или плохо структурированным *(ill structured)* относят такие задачи, которые обладают одной или несколькими из следующих характеристик:

1. Задачи не могут быть заданы в числовой форме (данные представлены в символьном виде).
2. Не существует точного метода решения.
3. Есть возможность выбора, но нет алгоритмического решения.
4. Противоречивость исходных данных и знаний о предметной области.

К предпосылкам появления и развития логического программирования можно отнести следующие недостатки традиционного подхода:

1. Некоторые понятия *плохо формализуются* с помощью алгоритмов.
2. Не все виды понятий можно представить программой.
3. Некоторые понятия и отдельные запросы определены частично или не полностью.
4. Программы теряют способность к модификации с увеличением их размеров.
5. Программы не могут определять или доопределять понятия.
6. Программы выводят только запланированные результаты и/или обрабатывают ошибочные ситуации, но альтернативные решения не предусматриваются.

Решение каждой из этих проблем требует перепрограммирования, что по стоимости часто равносильно созданию новых программ. Поэтому

представляет интерес иной подход (в отличие от традиционного) к решению запросов пользователя.

Логическое программирование позволяет учитывать главную проблему – использование компьютера силами самого пользователя, не обладающего знаниями в области программирования, но хорошо ориентирующегося в своих прикладных областях, и владеющего профессиональными специфическими знаниями.

Логическое программирование учитывает также и возможности, присущие традиционному процедурному программированию.

1.2. Историческая справка о языке Пролог

Одним из самых знаменитых языков логического программирования является *Пролог*, название которого произошло от словосочетания **ПРО***граммирование при помощи* **ЛОГ***ики (***PRO***gramming in* **LOG***ic)*.

В настоящее время *Пролог* – язык, предназначенный для создания приложений, использующих методы и средства искусственного интеллекта, и для создания экспертных систем.

Среди *отцов Пролога* называют ведущих учёных: *А. Колмероэ, Дж. Робинсона и Р. Ковальски*.

Алан Колмероэ – непосредственно автор *Пролога* как языка. В 1971 году в "Группе искусственного интеллекта" Марсельского университета во Франции родился *Пролог*. Главной задачей группы было создание программ для перевода с естественного языка. А. Колмероэ решил использовать для задач понимания и анализа естественного языка две области науки:

- логику предикатов первого порядка;
- методику автоматического доказательства теорем.

Суть языка в том, чтобы основываясь на определённых отношениях между объектами, путём логического вывода доказать истинность (или ложность) других отношений.

Первое время (в начале 70-х годов XX века) *Пролог* был не очень популярен, он пребывал в некоем забвении, вызванном отсутствием хороших реализаций, и был известен в основном в университетских кругах Франции.

Вторым столпом называют *Робинсона Дж.*, предложившего метод резолюций для поиска решений. Коротко можно сказать об этом методе следующее: получается ответ "НЕТ", если невозможно найти ответ "ДА".

Третьим патриархом логического программирования стал *Роберт Ковальски* из шотландского университета в Эдинбурге. Основные заслуги Р. Ковальски: состоят в следующем:

- он первый подвёл стройную математическую и логическую базу под язык;
- разработал формальную семантику языка;
- явился первым и наиболее энергичным популяризатором языка на конгрессах, конференциях, в литературе и т.п.

Математическая модель языка *Пролог* основана на теории исчисления предикатов, в частности, на процедурной интерпретации дизъюнктов Хорна. Модель в алгоритмическом (машинно-ориентированном) виде выразил *Маартен ван Эмден* – коллега Р. Ковальски.

Первый интерпретатор *Пролога* был написан *Ф. Русселом* в 1973 году в лаборатории А. Колмероэ. Однако первой эффективной и реальной (т.е. коммерческой) реализацией языка является так называемая эдинбургская версия *Дэвида Уоррена* для машин класса DEC-10.

Можно выделить три фактора, позволивших *Прологу* в начале 80-х годов XX века выйти на мировую арену вычислительной техники из *университетского подполья*:

1. Заинтересованность фирмы DEC в реализации нового языка, и появление первой коммерческой версии.
2. Математическое обоснование логического базиса языка.
3. Объявление языка *Пролог* в качестве базового в японском проекте создания вычислительных систем 5-го поколения под названием *FGCS (Fifth Generation Computer Systems).*

Японский проект, рассчитанный на десять лет, стартовал в апреле 1982 года под эгидой Токийской международной конференции. Разработка проекта осуществлялась в исследовательском центре *ICOT (The Institute for New Generation Computer Technology)* Токийского института нового поколения вычислительной техники под руководством *Кацухиро Фучи*. Проект был завершён в марте 1993 года.

Цель проекта – выполнение основных исследований по разработке новой технологии компьютеров, обусловленных необходимостью создания настолько совершенных машин, чтобы с ними легко могли работать самые широкие круги пользователей. Такие машины должны обладать способностями к приобретению знаний, самообучению, рассуждениям и решению задач. Взаимодействие машин с человеком должно быть ориентировано на человека.

Японцы изначально открыто заявили, что *FGCS* не носит коммерческой направленности, и результатом его будет не какой-то конкретный

программный продукт, а новые компьютерные технологии. Проект создания вычислительных систем 5-го поколения был чисто исследовательским, более того фундаментальным.

По мнению разработчиков проекта, логическое программирование способно стать связующим звеном, позволяющим с единых позиций взглянуть на современные тенденции развития по следующим направлениям:

- программирование,
- системы баз данных,
- архитектура компьютеров,
- искусственный интеллект.

Согласно проекту, основной архитектурной единицей вычислительной системы должна была быть машина логического вывода с языком логического программирования *KL1* (расширенный *Пролог*) в качестве машинного. На первом этапе – последовательная персональная машина логического вывода, затем – параллельная система (от 256 до 1024 процессоров).

Все прикладные системы, написанные на *KL1*, работают со скоростью, почти линейно пропорциональной числу процессоров (два процессора – в два раза быстрее, четыре процессора – в четыре раза быстрее и т.д.). Например, решатель теорем работает на машине с 256 процессорами в 220 раз быстрее, чем на однопроцессорном компьютере! До сих пор подобного соотношения на компьютерах, созданных в США и Европе, удавалось добиваться только для единичных приложений, создаваемых специально под конкретную архитектуру. Реальный выигрыш во времени (ускорение выполнения программы) для неяпонских параллельных компьютеров обычно составляет для 300-процессорноых машин – до 30 раз, для машин с числом процессоров более тысячи – до 250 раз.

В середине 1995 года в связи с завершением *FGCS* был отобран ряд наиболее перспективных направлений, которые относятся как к чисто научным исследованиям, так и к конкретным прикладным задачам. Эти направления развиваются в стенах японского НИИ современных информационных технологий *AITEC (Research Institute for Advanced Information Technology)*. Наиболее перспективными признаны следующие области логического программирования:

- технологии интеллектуальной параллельной обработки знаний,
- языки баз знаний на основе *Пролог*-подобных систем,
- решатели интеллектуальных задач,
- программы анализа и обработки естественных языков,
- базы знаний по биологии.

Базовая идея, лежащая в основе японского проекта – создание высокоэффективных компьютерных средств обработки знаний. Понятие *знание* японскими специалистами в компьютерном контексте трактуется как *"формализованная информация, которую используют и на которую ссылаются в процессе логического вывода"*.

Компьютерные знания в логическом программировании принято делить на факты, описывающие окружающий мир, и на правила, которые позволяют программе принимать различные решения в процессе логического вывода.

2. ОСНОВНЫЕ ПОНЯТИЯ ЯЗЫКА ПРОЛОГ

Пролог является декларативным языком. Программист задаёт необходимые факты и правила, а *Пролог* использует дедуктивный вывод для решения задачи. Программист имеет возможность заниматься непосредственно задачей, а не поиском способа разделения её решения на небольшие шаги. Такой метод представляет собой полную противоположность программированию на каком-либо процедурном языке, где решение разбивается на дискретные шаги и их последовательное описание.

Единственным методом программирования на *Прологе* является рекурсия.

В сущности, после завершения работы программы на *Прологе* (доказательства цели) могут возникнуть только два состояния: *доказано, не доказано*. Иногда они называются соответственно *истинно* и *ложно*.

Необходимо научиться формулировать логические соотношения, описывающие задачу, а *Пролог*-система выполнит все необходимые действия.

Есть две основные абстракции логического программирования:

1. Отказ от таких операторов, как хорошо известные операторы цикла и условные операторы. Вместо них мощный механизм управления, который единообразно применяется во всей программе. Механизм основан на понятии доказательства в математической логике: программа рассматривается как набор логических формул (аксиом), которые предполагаются истинными, а вычисление – как попытка доказать формулу (теорему) на основе аксиом программы.
2. Не используются операторы присваивания и явные указатели. Вместо этого для создания и декомпозиции структур данных используется обобщённый механизм сопоставления с образцом.

2.1. Структура программы на языке Пролог

Основным компонентом *пролог*-программы является предложение, которое может быть фактом, правилом или целью. В табл. 2 приведены примеры предложений, из которых строится логическая программа.

Таблица 2

Предложения программы на языке Пролог

Предложение	Пролог	Естественный язык
факт	студент (иван).	Иван является студентом
правило	ходит (**X**, лекция) **if** студент (**X**).	Если **X** студент, то **X** ходит на лекцию
цель	ходит (**иван**, лекция).	Иван ходит на лекцию?

иван – конкретный объект, **X** – абстрактный объект.

Пример 2-1. Простейшая пролог-программа.

```
predicates
        nondeterm нравится (symbol, symbol)
clauses
        нравится (лена, теннис).
        нравится (петя, футбол).
        нравится (коля, баскетбол).
        нравится (женя, плавание).
        нравится (марк, теннис).
        % Егору нравится то же, что и Коле.
        нравится (егор, X) if
                нравится (коля, X).
goal    нравится (егор, баскетбол).        % Нравится Егору баскетбол?
```

Любая программа на *Прологе* состоит из пяти секций или разделов, некоторые из которых могут быть опущены. В *Примере 1* используются три из них, обозначенные ключевыми словами **predicates**, **clauses**, **goal**.

Секция **clauses** *(пролог-предложения или клозы)* содержит набор фактов и правил. Среди фактов нет информации о том, что нравится ли Егору баскетбол. Тем не менее, этот вопрос можно задать системе в виде цели **goal**. На ввод этой цели будет получен результат **yes**.

Для получения ответа система использует один факт и одно правило:

```
(1) нравится (коля, баскетбол).
(2) нравится (егор, X) if нравится (коля, X).
```

Предложение (1) – *факт*, где в отношении **нравится** участвуют два конкретных объекта **коля** и **баскетбол**.

Предложение (2) – *правило*, которое отличается от факта наличием абстрактного объекта **X** и некоторого условия, от выполнения которого зависит истинность этого правила.

Если заменить целевое утверждение другим:

```
goal    нравится (егор, плавание).
```

Будет получен результат **no**. Этот ответ не означает, что *Егору не нравится плавание*. Ответ **no** означает, что системе *не известен факт* любви Коли к плаванию.

Ответ **yes** на последнюю цель можно получить, если:

- добавить факт **нравится (коля, плавание).**
- или добавить факт **нравится (егор, плавание).**

Таким образом, программа на *Прологе* описывает объекты и их связи, а затем описывает правила, при которых связи являются истинными.

Пролог-программа состоит из предложений трёх типов: *факты, правила и цели*. Каждое предложение заканчивается точкой.

Факт – это утверждение, которое всегда является истинным без всякого условия.

В фактах используются только объекты–константы.

Правило – это утверждение, истинность которого доказывается в зависимости от заданного условия.

Правило позволяет выводить один элемент информации из другого благодаря наличию в нём переменных. Правило успешно, только если достижимы заданные в теле правила цели, иначе правило терпит неудачу.

Цель – это утверждение, которое задают как вопрос относительно фактов и правил.

В целях применяются переменные, а также константы.

2.2. Назначение секций Пролог-программы

Любая программа состоит секций (табл. 3), которые обозначаются ключевыми словами **domains**, **facts**, **database**, **predicates**, **goal**, **clauses**. При отсутствии секций **predicates**, **goal** и **clauses** *Пролог*-система не будет выполнять программу. Остальные секции могут быть опущены.

Таблица 3

Секции Пролог-программы

Ключевое слово	Назначение	Пояснения
domains	Описание доменов	Содержит объявления различных классов используемых в программе объектов.
facts	Описание фактов	Содержит объявления *недетерминированных предикатов,* представленных в программе фактами. Такие предикаты можно объявлять и в секции **predicates**, тогда секции **facts** не будет.

Продолжение табл. 3

Ключевое слово	Назначение	Пояснения
database	Описание предикатов ДБД	Содержит объявления предикатов динамической базы данных (ДБД). Если программа не требует ДБД, то секция может быть опущена.
goal	Целевое утверждение	Является обязательной секцией. Содержит формулировку назначения программы в виде цели. **Цель** должна быть только **одна**.
predicates	Описание отношений (кроме ДБД)	Является обязательной секцией. Содержит объявления используемых программой предикатов (фактов, правил). По умолчанию все предикаты считаются *детерминированными*. Если предикат порождает множество решений, то перед его именем следует указать ключевое слово **nondeterm**.
clauses	Утверждения	Является обязательной секцией. Содержит известные априорно факты и правила, которые являются необходимыми данными для работы программы. Предложения группируются по именам предикатов. Порядок следования предложений неважен, но зачастую влияет на процесс поиска решения. Факты, объявленные в секции **predicates** (но не в секции **facts**) рассматриваются как статическая база данных.

2.3. Константы и переменные

Константа обозначает конкретный объект и определяет ***одну конкретную сущность*** *предметной области.*

Константа в *Прологе* может быть числом или атомом.

Атом – последовательность заключённых в кавычки символов *или* последовательность букв, цифр и символов подчёркивания, которая начинается со строчной буквы. Атомы без кавычек не могут содержать пробелов.

Примеры констант:

17 2.5 "Daniel H. Marcellus" logic "Санкт-Петербург"

Понятие переменной в логическом программировании отличается от принятого в традиционных языках программирования. В частности, переменная не рассматривается как выделенный участок памяти, которому

присвоено имя. В программе переменная обозначается последовательностью символов, которая начинается с прописной буквы.

Переменная обозначает ***неопределённый (но единственный объект)*** *и служит для обозначения объекта, на который нельзя сослаться по имени.*

Переменная может быть свободной, связанной и анонимной.

Свободная переменная – это переменная, значение которой не известно.

Связанная переменная – это переменная, получившая значение в процессе логического вывода.

Начинается переменная с *прописной* буквы или символа *подчёркивания*, например,

What Who Что X _X _x _кто

Процесс получения значения переменной называется конкретизацией.

Анонимная переменная – это переменная, используемая для игнорирования значения, в котором нет нужды.

Анонимная переменная может использоваться вместо любой другой переменной. Она соответствует чему-нибудь. Анонимная переменная никогда не будет связываться с конкретным значением. В программе такая переменная обозначается *символом подчёркивания*.

2.4. Объекты и отношения

Объекты логического программирования представляют собой элементы предметной области и могут быть:

- конкретными объектами (атомами),
- абстрактными объектами (переменными).

Объекты взаимодействуют друг с другом, то есть, между ними существует связь, которая их объединяет.

Набор однотипных связей между объектами называют отношением.

В *Примере 1* два объекта **марк** и **теннис** взаимодействуют в отношении **нравится**:

нравится

марк → **теннис**

Конкретный объект обладает вполне определённым свойством или свойствами. В логических программах не существует различий между отношением и свойством. Термин *отношение* применяется и к свойствам тоже.

Примеры свойств:

car
→ mercedes, blue

red
→ nose

Объекты с отношениями являются неотъемлемой частью данных и образуют утверждения предметной области. На первом месте в утверждении указывается имя отношения, а в скобках через запятую – объекты отношения. В конце обязательно ставится точка.

Примеры утверждений:

car (mercedes, blue).	**red (nose).**
likes (tom, ann).	**plus (3, 2, 5).**
girl (mary).	**run.**

Очевидно, что утверждение может содержать разное количество объектов.

Для обозначения числа объектов утверждения введён термин *арность*.

Утверждения **car** и **likes** двух-арные, **red** и **girl** – одно-арные, **plus** – трёх-арное, **run** – ноль-арное.

Запись отношения позиционно зависимая, т. е. **plus (3, 2, 5).** НЕ РАВНО **plus (5, 3, 2).**

В общем виде факты имеют следующий формат:

relation (object$_1$, object$_2$, ..., object$_N$) .
property (object$_1$, object$_2$, ..., object$_N$) .
property (object$_1$) .

, где **object$_j$** – объекты, связанные отношением **relation** или свойством **property**.

Примеры фактов:

isA (susan, horse).	**eats (bill, apple).**
power (knowlege).	**car (mercedes, blue, berlin).**

2.5. Предикаты

Интуитивно понятно, что каждое отношение имеет некоторую интерпретацию. Например, известно:

*Студенческая практика имеет определённое название **(Имя)** и проводится на определённом курсе **(Курс)**, кроме того, нет двух студенческих практик с одинаковыми названиями.*

Для отношения **назначитьПрактику** между объектами **Имя** и **Курс** могут существовать следующие интерпретации:

назначитьПрактику (учебная, 1).
назначитьПрактику (производственная, 3).
назначитьПрактику (исследовательская, 4).

Формально, предложенные утверждения – это пример функции значения истинности аргументов (в данном случае двух аргументов).

Предикат – это логическая функция истинности, которая для некоторого аргумента возвращает значение истинно или ложно.

Предикат, содержащий **n** аргументов, называется **n-**арным (-местным) предикатом.

Подстановка значений аргументов предиката эквивалентна вызову функции, а значит, получению высказывания, которое может быть истиной либо ложью.

Так, при подстановке **Имя = производственная** и **Курс = 3**

получается *истинное высказывание*

назначитьПрактику (производственная, 3).

При подстановке **Имя = исследовательская** и **Курс = 3**

получается *ложное высказывание*

назначитьПрактику (исследовательская, 3).

Последнее высказывание ложно, потому что на третьем курсе студенты проходят не исследовательскую, а производственную практику.

Символическое имя отношения называется именем предиката.

Объекты, которые связаны отношением, называются его параметрами.

Предикат в Прологе – это символическое имя отношения и последовательность параметров.

Например, в факте **eats (ted, apple).**

отношение **eats** – предикат, а объекты **ted** и **apple** – параметры.

2.6. Области определения предикатов и объектов

Все предикаты, используемые в программе, должны быть объявлены согласно формату следующего вида:

$$\textbf{PredicateName (argType}_1\textbf{, argType}_2\textbf{, ..., argType}_N\textbf{)}$$

, где **PredicateName** – имя предиката,
argType$_j$ – тип аргумента.

Следует обратить внимание, что в конце точка не ставится, и параметры предиката в секции **predicates** *не являются объектами*, а представляют собой типы объектов, входящих в предикат.

Для того чтобы задать системе вид предиката и его арность, в программе используется секция **predicates**. Например:

```
predicates
        eats (symbol, symbol)
```

Предикаты, определённые стандартными типами объектов, называются типовыми.

Пролог имеет шесть встроенных типов объектов, которые представлены в табл. 4.

Таблица 4

Встроенные типы объектов

№	Ключевое слово	Тип	Пояснения		
1	**char**	Символы	'a'	'%'	'\n'
2	**integer**	Целые числа	-32768		32767
3	**real**	Действительные числа	1E-307		1E308
4	**string**	Строки	Последовательность символов между " " длиной менее или равной 250 символов		
5	**symbol**	Символические имена	Атом		Аналогично типу **string**
6	**file**	Файлы	Допустимое имя файла: data.dba, mail.txt		

Определение типов происходит в специальной секции программы **domains**. Формат объявления объектов:

$domainName_1$, $domainName_2$, …, $domainName_N$ = type

, где **$domainName_j$** – доменные имена типов объектов,
type – стандартный встроенный тип объектов.

Пример 2-2. Использования в программе объектов разных типов.

```
domains
        brand, color = symbol
        age = integer
        mileage = real
predicates
        car (brand, mileage, age, color)
```

Использование доменных имён при объявлении предикатов в логической программе наиболее предпочтительно в сравнении с использованием стандартных встроенных типов объектов.

2.7. Цели

Вопрос, который задают относительно фактов или правил пролог-программы, называют целевыми утверждениями, или просто целями.

Цели задаются в секции **goal**.

Различаются простые и сложные цели. *Простая цель* состоит из одного предложения. *Сложная цель* – из нескольких предложений, соединённых логической операцией *конъюнкция*.

В конце целевого предложения обязательно ставится *точка*.

В зависимости от используемых объектов различают варианты целей:

1. Простая цель с константами.
2. Простая цель с переменными.
3. Сложная цель.
4. Цель с анонимными переменными.

Разные цели исследуются на примере следующей программы:

```
domains
        brand, color = symbol
        age = integer           mileage = real
predicates
        car (brand, mileage, age, color)
clauses
        car (volvo, 13000, 3, red).
        car (ford, 9000, 4, gray).
        car (datsun, 8000, 1, red).
```

2.7.1. Простая цель с константами

В качестве цели задан предикат, все параметры которого представлены константами.

Основной результат работы программы – подтвердить или не подтвердить истинность указанного в целевой секции предиката.

```
goal        car (ford, 9000, 4, gray).
yes
goal        car (volvo, 9000, 4, gray).
no
```

2.7.2. Простая цель с переменными

В качестве цели подставлен предикат, в котором есть переменные.

Пролог-система будет стараться найти **все** такие **значения переменных**, при которых *целевой предикат становится истинным*.

```
goal        car (volvo, 13000, Age, Color).
Age = 3     Color = red
1 Solution
```

```
goal        car (Brand, Mile, Age, red).
Brand = volvo          Mile = 13000          Age = 3
Brand = datsun         Mile = 8000           Age = 1
2 Solutions
```

2.7.3. Сложная цель

Целевое утверждение состоит из двух или более частей, называемых подцелями. Подцели соединяются, как правило, логической связкой **И**.

В этом случае *Пролог* будет искать **все** такие **значения переменных**, при которых *все подцели будут истинными*.

```
goal
        % Есть ли автомобили возрастом меньше 4 лет?
        car (Type, Odometer, YearOnRoad, Color) and YearOnRoad < 4.
Type = volvo     Odometer = 13000     YearOnRoad = 3  Color = red
Type = datsun    Odometer = 8000      YearOnRoad = 1  Color = red
2 Solutions
```

2.7.4. Цель с анонимными переменными

Если необходимо скрыть какую-либо информацию в запросе, то следует воспользоваться анонимной переменной.

Анонимная переменная используется для игнорирования значения, в котором нет нужды.

В *Прологе* анонимная переменная обозначается символом подчёркивания.

```
goal
% Каковы марки и возраст автомобилей, находящихся на дорогах < 4 лет?
        car ( Type, _, YearOnRoad, _ ) and YearOnRoad < 4.
Type = volvo     YearOnRoad = 3
Type = datsun    YearOnRoad = 1
2 Solutions
```

Пример лабораторной работы № 1 приведён в [29].

2.8. Несколько пояснений по синтаксису программ на Прологе

Многострочные комментарии записываются в программе аналогично языку программирования C:

```
/* Многострочный комментарий */
```

Однострочный комментарий начинается с символа *процент*:

```
% Однострочный комментарий
```

Для обеспечения более лёгкого чтения текстов программ, следующие слова и символы являются попарно заменяемыми:

if	эквивалентно	**:–**	*(двоеточие и минус)*
and	эквивалентно	**,**	*(запятая)*

Таким образом, два следующих фрагмента идентичны:

```
isOlder (P1, P2)   if
      age (P1, Age1)   and
      age (P2, Age2)   and
      Age1 > Age2.
```

```
isOlder (P1, P2)   :–
      age (P1, Age1),
      age (P2, Age2),
      Age1 > Age2.
```

Все предложения, представляющие факты и правила, обязательно заканчиваются символом *точка*.

Возможно описание нескольких правил для одного и того же предиката, что соответствует логической связке **OR (ИЛИ)**.