

Т.В. АВДЕЕНКО, М.Ю. ЦЕЛЕБРОВСКАЯ

# ВВЕДЕНИЕ В ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ ПРОГРАММИРОВАНИЕ В СРЕДЕ VISUAL PROLOG

Утверждено Редакционно-издательским советом университета  
в качестве учебного пособия

НОВОСИБИРСК  
2020

УДК 004.8(075.8)  
А 187

Рецензенты:

*М.Г. Зайцев*, канд. физ.-мат. наук, доцент  
*М.Ш. Муртазина*, канд. филос. наук, доцент

Работа подготовлена на кафедре ТПИ для студентов III курса ФПМИ  
(направления подготовки 01.03.02, 02.03.03)

**Авдеенко Т.В.**

А 187 Введение в искусственный интеллект и логическое программирование. Программирование в среде Visual Prolog: учебное пособие / Т.В. Авдеенко, М.Ю. Целебровская. – Новосибирск: Изд-во НГТУ, 2020. – 64 с.

ISBN 978-5-7782-4182-4

Настоящее учебное пособие представляет собой вводную часть курса по искусственному интеллекту, основной целью которого является изучение модели представления знаний на основе классических логических исчислений – исчисления высказываний и исчисления предикатов. Пособие затрагивает не только теоретические основы рассматриваемой модели представления знаний, но и ее реализацию на языке логического программирования Пролог. Таким образом, студенты не только овладевают теоретическими основами представления знаний в логической модели, но и получают практические навыки применения знаний при написании и отладке логических программ.

УДК 004.8(075.8)

ISBN 978-5-7782-4182-4

© Авдеенко Т.В., Целебровская М.Ю., 2020  
© Новосибирский государственный  
технический университет, 2020

## **1. ОБЩИЕ СВЕДЕНИЯ О ЯЗЫКЕ ПРОГРАММИРОВАНИЯ ПРОЛОГ**

В отличие от подавляющего большинства других языков программирования Пролог (Prolog) обычно рассматривается в одном контексте с понятием «логическое программирование», которое, в свою очередь, является промежуточным этапом развития языков программирования в направлении ухода от «процедурности» ко все большей «декларативности», что соответствует основной идее развития методологии искусственного интеллекта (ИИ) [8]. Вероятно, в скором будущем человек, решая задачу, не будет задавать последовательность команд на некотором известном языке программирования (чисто процедурный подход), а станет описывать ее в совершенно абстрактных логических терминах, не оперирующих определениями «байт» или «указатель», т. е. он будет создавать модель анализируемой проблемы и пытаться получить положительные или отрицательные результаты этого анализа. Практическим результатом такого анализа может быть, например, автоматически генерируемая на основе созданной модели оптимизированная программа на процедурном языке (например, C++).

Таким образом, можно констатировать, что Пролог не является языком программирования в чистом виде. С одной стороны, этот язык можно рассматривать как оболочку экспертной системы (ЭС), с другой – как интеллектуальную базу данных и, что самое важное, не реляционную. Математическая модель, лежащая в основе Пролога, довольно сложна, и по мощности системы формирования запросов к базе с этим языком не сравнится ни одна из коммерческих СУБД.

Из сказанного следует, что Пролог является не процедурным, а декларативным языком программирования и относится к группе постобъектно-ориентированных языков – функциональным языкам. Человек лишь описывает структуру задачи, а Пролог-машина вывода сама ищет

решение. Более того, здесь вообще не существует понятия последовательности команд, все это скрыто в математической модели языка. Однако заметим, что, хотя в идеальной модели Пролог-машины последовательность целей не играет роли, в ее конкретных реализациях от порядка следования целей зависит не только скорость работы программы, но часто и получаемый результат.

Математическая модель Пролога основана на теории исчисления предикатов, в частности на процедурной интерпретации хорновых дизъюнктов, содержащих не более одного заключения, предложенной Робертом Ковальским из Эдинбурга [7]. Алан Колмероз, автор языка Пролог, начал работы над полноценной компьютерной реализацией трудов Ковальского с 1972 года. Он составил алгоритм формального способа интерпретации процесса логического вывода и разработал систему автоматического доказательства теорем, которая была написана на Фортране. Она-то и послужила прообразом Пролога. Название «Пролог» произошло от *Programmation en Logique* – ЛОГическое ПРОграммирование. Вскоре появились первые компиляторы с этого языка, в частности прекрасная реализация Дэвида Уоррена для компьютера DEC-10 в Эдинбурге, ставшая своего рода стандартом вплоть до сегодняшнего дня. Эффективность этой версии заставила специалистов по искусственному интеллекту по-новому взглянуть на Пролог. В некоторых приложениях, типичных для Лиспа, таких как обработка списков, Пролог уже не уступал своему конкуренту, что и послужило в дальнейшем стимулом для ряда специалистов по логическому программированию к переходу на этот язык.

В качестве типовых данных Пролог использует элементарные единицы данных, так называемые атомы – строки символов и числа. Из атомов составляются списки и бинарные деревья. Сама «программа» строится из последовательности фактов и правил, и затем формулируется утверждение, которое Пролог будет пытаться доказать с помощью введенных правил. Таким способом можно описывать очень сложные проблемы, которые будут решаться Прологом *автоматически*, т. е. алгоритм решения задачи (последовательность действий) строится автоматически в процессе интерпретации логической программы. Это происходит с помощью метода сопоставления и рекурсивного поиска. Рекурсия играет в Прологе большую роль.

С распространением Пролога появилась возможность создавать интеллектуальные нереляционные базы знаний с иерархической структурой на основе стандартного механизма с гибкой организацией очень

сложных запросов. Были написаны эффективные программы для решения переборных задач, в частности из области молекулярной биологии и проектирования СБИС, где требовалось учитывать либо сложные внутренние структуры, либо большое число правил, описывающих организацию объекта. Пролог хорошо зарекомендовал себя в качестве экспертной оболочки и при решении задач грамматического разбора. Что весьма характерно, первый высокопроизводительный компилятор этого языка (Эдинбургская версия) был написан на самом Прологе. И немудрено, ведь все формальные синтаксические описания грамматик в Бэкус-форме прекрасно записываются в терминах Пролога.

Долгое время среди разработчиков этого языка шла напряженная борьба между сторонниками оригинальной семантики Пролога и специалистами, стремившимися пожертвовать ясной структурой языка ради повышения эффективности реализации. В частности, свою роль стала играть последовательность правил в базе данных. Дело в том, что нередко для получения быстрого ответа целесообразно использовать сначала, например, наиболее простые правила или наиболее эффективные с точки зрения человека. Программа на Прологе постепенно стала приближаться к обычным процедурным языкам. Немалую роль в этом сыграло и искусственно введенное понятие отсечения (реализуемое с помощью предиката «!»), своего рода аналог столь нелюбимого профессором Дейкстром оператора «go to». Теперь программист мог по своему усмотрению динамически отсекал бесплодные (по его мнению) ветви деревьев перебора, что приводило к многократному (на два-три порядка) повышению скорости работы программ, но при этом существенно нарушалась ясность ее структуры, и возникновению множества проблем, связанных с отладкой.

К счастью, развитие вычислительной техники в сочетании с уникальной структурой языка дало свои результаты. При появлении первых параллельных компьютеров программисты, работающие на Прологе, быстро осознали пагубность различных «нововведений» типа оператора отсечения, лишавших язык оригинальной чистоты, и вернулись к первоначальной версии языка. Пресловутая последовательность правил перестала играть роль, так как появилась возможность вычислять их параллельно, а поскольку математическая теория Пролога не накладывает никаких требований на упорядоченность фактов и правил в базе, скорость работы программы стала линейно пропорциональной числу процессоров. Имеются бесплатные и коммерческие версии Пролога для реализаций на параллельных компьютерах, например, Densitron CS Prolog

для транспьютеров, или Parallogic. В японском проекте компьютера пятого поколения все программное обеспечение создавалось на Пролог-подобном языке.

Большинство свободно распространяемых версий Пролога для обычных однопроцессорных компьютеров сегодня или является усеченными подмножествами коммерческих продуктов, или поставляется бесплатно только для учебных и научных организаций.

Перечислим некоторые из них: SWI-Prolog, Amzi!-Prolog. Знакомый многим программистам TurboProlog, разрабатывавшийся ранее фирмой Borland, теперь выступает под маркой PDC Prolog и реализован для DOS, Windows, OS/2 и UNIX. Правда, как недостатки (например, несовместимость с подавляющим большинством других диалектов этого языка), так и его достоинства (такие как удобная среда разработчика и быстрый компилятор) сохранились. Есть специальная версия для визуальной разработки Visual Prolog. На нем написана маленькая, простая и удобная Пролог-машина Prolog-Interface-Engine-PIE32. Именно на Visual Prolog рекомендуется выполнять практические задания, предлагаемые в настоящем учебном пособии.

Современные профессиональные Пролог-системы обеспечивают скорость работы, не уступающую скорости выполнения аналогичных программ, написанных на Си (конечно, если не решать на Прологе задачи, подобные Фурье-преобразованию).

В силу всех изложенных выше причин для знакомства студентов с функциональными языками рекомендуется язык Пролог. Общеизвестно, что это наиболее простой для изучения язык и в то же время очень мощный инструмент для построения интеллектуальных систем.

Все исследователи практически единогласно утверждают, что будущее – за функциональными языками. Примеры и задачи, описанные в учебном пособии, позволят студентам ознакомиться с наиболее перспективным на сегодняшний день направлением информатики, изучающим принципы функционирования и построения систем искусственного интеллекта.

## **1.1. ВВЕДЕНИЕ В ПРОЛОГ**

Как было сказано выше, теоретической основой Пролога послужил раздел математической логики, называемый исчислением предикатов. Прологу присущ ряд свойств, которыми не обладают традиционные языки программирования:

- механизм вывода с поиском и возвратом,

- встроенный механизм сопоставления с образцом,
- простая структура представления данных с возможностью их динамического изменения.

Процесс разработки программы на языке Пролог представляет для программиста запись знаний о предметной области в виде фактов и правил в терминах языка и формулировку цели-вопроса, позволяющего после выполнения программы получить ответ на него в форме «доказано – не доказано». В отличие от традиционных процедурных языков программирования Пролог не дает возможности записывать в программе последовательность команд, обязательных для исполнения. Роль исполнителя со стандартными алгоритмами работы берет на себя встроенный механизм Пролога, который в зарубежной литературе называется Engine – движок Пролога (интерпретатор логической программы). Ниже мы рассмотрим алгоритмы работы движка, так как четкое понимание этих алгоритмов является залогом создания правильно и эффективно работающей программы на Прологе.

После завершения работы Пролог-программы (доказательства цели) могут возникнуть только два состояния: «доказано – не доказано» [6]. Иногда они называются состояниями «Истинно» и «Ложно». Доказанное утверждение обычно является истинным, но это не обязательно, так как доказательство зависит от известных фактов и сделанных на их основе выводов. Итак, введены новые термины.

**1. Факт** – утверждение, которое определяется как доказанное (всегда истинное). Например, факт «Мухтар – это собака» считается доказанным по определению. Но ответ на вопрос «Является ли Джек собакой?» не будет дан, так как утверждение «Джек – собака» не может быть доказано с помощью известных фактов. Отрицательный ответ системы Пролог означает вовсе не то, что утверждение «Джек – собака» ложно, а лишь то, что в Пролог-программе нет фактов о том, что Джек является собакой. Такой вид отрицания в Прологе называется отрицанием через неуспех.

**2. Вывод.** Имея множество фактов, мы можем определять новые свойства описанных в них объектов, т. е. допускается вывод свойств из фактов.

Вернемся к факту «Мухтар – это собака». Предположим, что мы хотим на основе известных фактов выявить все объекты, обладающие свойством «быть собакой». Задав вопрос «Кто является собакой?», получим ответ «Мухтар – это собака». Это тривиальный пример вывода. Введем некоторую дополнительную информацию:

*Мухтар – это собака /\*факт\*/*  
*Блонди является родителем Мухтара /\*факт\*/*  
*Джек является родителем Мухтара /\*факт\*/*  
*Субъект является собакой при условии, что он является родителем собаки. /\*правило\*/*

Первые три элемента данных представляют собой факты (утверждения, которые всегда истинны), четвертый элемент представляет собой правило (импликация – утверждение, которое задает истинность заключения при условии истинности предпосылки). Теперь, если мы спросим: «Какие объекты являются собаками?», то получим ответ: «Мухтар, Блонди, Джек», причем Мухтар является собакой по определению, а Блонди и Джек – вследствие логического вывода (более точно – вследствие однократного применения правила *Modus Ponens* к правилу и соответствующему факту). В этом заключается основное различие между алгоритмическим и логическим мышлением. Если сформулировать приведенный выше запрос в алгоритмических терминах, то получится весьма сложное определение:

*Объект является собакой в том случае, если он имеет свойство «быть собакой» или же имеет свойство «быть родителем» и существует объект, связанный с родителем и обладающий свойством «быть собакой».*

Самый большой недостаток языка Пролог при этом заключается в том, что мы должны заранее в точности знать, какие будут заданы вопросы, и запрограммировать все необходимые факты и правила, которые будут давать на них ответы. Для того чтобы представить, как решается на Прологе задача «Какие объекты являются собаками?», приведем заготовку для программы на Прологе:

*собака (мухтар). /\*факт\*/*  
*родитель (блонди, мухтар). /\*факт\*/*  
*родитель(джек, мухтар). /\*факт\*/*  
*собака (X):-родитель(X,Y), собака(Y). /\*правило\*/*

В программе определяются свойства «быть собакой» и «быть родителем». Теперь, если мы хотим узнать, какие объекты являются собаками, то должны задать вопрос (поставить цель поиска для движка Пролога):

*собака (Кто).*



В данной программе, согласно правилам синтаксиса Пролога, элемент, начинающийся со строчной буквы, представляет собой константу, обозначающую объект или его свойство, и не может быть изменен в процессе выполнения программы. Слово, начинающееся с прописной буквы, является переменной. Во время выполнения программы переменная может быть сопоставлена (унифицирована) с определенным значением.

Запрос *собака (Кто)* выполняется следующим образом.

Движок Пролога просматривает программу с начала и ищет ответ на вопрос «Какое значение должна принять переменная Кто, если мы хотим доказать названную выше цель?». Проследивая программу с начала, мы видим, что, если переменная *Кто* получит значение «мухтар», то целевое утверждение будет согласовано. Ответ программы будет выглядеть следующим образом:

*Кто = мухтар.*

Затем движок Пролога начнет искать другие решения. Для того чтобы найти следующий ответ, движок запоминает, где был получен последний результат, и продолжает работу с этого места в поисках другого варианта решения. Переменная *Кто* при этом теряет значение «мухтар». Теперь предстоит сопоставление с утверждением, содержащим правило вывода.

Заметим, что переменная *Кто* сцеплена с пока свободной переменной *X*, стоящей в заголовке правила *собака (X)*. Сцепление переменных обозначает, что они сейчас размещены в виде пары в специальном рабочем для движка участке памяти, называемом стеком. Поэтому, когда переменная *X* принимает какое-либо значение, то и переменная *Кто* принимает это же значение.

Теперь мы подошли к утверждению «если»: «*Объект является собакой при условии, что он является родителем кого-то и этот кто-то является собакой*». На Прологе утверждение записывается следующим образом: за головной целью «*собака X*» следует символ «:-», означающий «если», а за ним – конъюнкция целей. Конъюнкция в Прологе обозначается символом «,». Так, «*родитель(X,Y),собака(Y)*» есть конъюнкция (логическое И) целей «*родитель(X,Y)*» и «*собака(Y)*». В Прологе существует также дизъюнкция, обозначаемая символом «;» и означающая «цель1 ИЛИ цель2», однако в больших рекурсивных программах не рекомендуется использовать «;» (операция «или»), так как она корректно действует лишь на верхнем уровне рекурсии.

Чтобы согласовать цель *собака* ( $X$ ), нужно согласовать цели *«родитель( $X,Y$ )»* и *«собака( $Y$ )»*.

При согласовании цели *«родитель( $X,Y$ )»* Пролог сопоставляет цель с первым фактом, относящимся к родителям, а именно – *«родитель(блонди,мухтар)»*. Переменные получают значения (унифицируются)  $X = \text{блонди}$ ,  $Y = \text{мухтар}$ . Поскольку  $X$  сцеплена с переменной *Кто*, переменная *Кто* тоже унифицируется (становится равной) значению *блонди*.

Унификации этими значениями переменных недостаточно, мы должны точно так же согласовать (унифицировать) цель *«собака( $Y$ )»*, которая теперь эквивалентна *«собака(мухтар)»*. Такое утверждение найдено, и цель *«родитель( $X,Y$ )»* согласуется при  $X = \text{блонди}$  и  $Y = \text{мухтар}$ . Таким образом, цель *«собака(Кто)»* повторно согласована при  $\text{Кто} = \text{блонди}$ .

Далее Пролог ищет другие альтернативные решения задачи, пытаясь согласовать самую последнюю цель *«собака(мухтар)»*. Однако такой возможности нет. Тогда Пролог пытается вновь согласовать предыдущую цель *«родитель( $X,Y$ )»*. Так как Пролог запомнил, что цель *«родитель( $X,Y$ )»* была согласована фактом *«родитель(блонди,мухтар)»*, он находит альтернативный факт – *«родитель(джек,мухтар)»*. Затем Пролог успешно производит согласование *«собака(мухтар)»*. Следовательно, найден альтернативный способ доказательства цели *«собака(Кто)»*:  $\text{Кто} = \text{Джек}$ .

Написанный нами проект программы позволяет Прологу найти ответы и на следующие вопросы.

1. Какие собаки имеют родителей?
2. Кто является родителем Мухтара?
3. Кто является родителем?

Для их обработки нет необходимости вносить изменения в программу. Мы просто вводим запросы (цели):

- 1) *родитель(\_,Потомок)*.
- 2) *родитель(Родитель,мухтар)*.
- 3) *родитель(Родитель,\_)*.

Символ « $\_$ » в целях означает так называемую анонимную переменную. Анонимная переменная используется тогда, когда для ответа на вопрос не важно, какое значение примет стоящая на этом месте переменная.

Далее, получив в работу эти цели, движок Пролога просматривает базу фактов и правил в соответствии с описанным выше примером и ищет необходимые решения.

## 1.2. УНИФИКАЦИЯ

Одним из важных аспектов программирования на Прологе являются понятия **унификации** (отождествления) и **конкретизации** переменных.

Пролог пытается отождествить термы при доказательстве или согласовании целевого утверждения. Например, в программе из раздела 1.1 для согласования запроса *собака (X)* целевое утверждение *собака (X)* было отождествлено (унифицировано) с фактом *собака (мухтар)*, в результате чего переменная  $X$  становится конкретизированной (унифицированной):  $X = \text{мухтар}$ .

Переменные, входящие в утверждения, отождествляются со своими значениями особым образом – сопоставляются. Факт доказывается для всех значений переменной (переменных). Правило доказывается для всех значений переменных в головном целевом утверждении при условии, что хвостовые целевые утверждения доказаны. Предполагается, что переменные в фактах и головных целевых утверждениях связаны квантором всеобщности. Переменные принимают конкретные значения на время доказательства целевого утверждения.

В том случае, когда переменные содержатся только в хвостовых целевых утверждениях, правило считается доказанным, если хвостовое целевое утверждение истинно для одного значения переменных или более. Переменные, содержащиеся только в хвостовых целевых утверждениях, связаны квантором существования. Таким образом, они принимают конкретные значения на то время, когда целевое утверждение, в котором переменные были согласованы, остается доказанным.

Терм  $X$  сопоставляется с термом  $Y$  по следующим правилам. Если  $X$  и  $Y$  – константы, то они сопоставимы только в том случае, если они одинаковы. Если  $X$  является константой или структурой, а  $Y$  – неконкретизированной переменной, то  $X$  и  $Y$  сопоставимы, и  $Y$  принимает значение  $X$  (и наоборот). Если  $X$  и  $Y$  – структуры, то они сопоставимы тогда и только тогда, когда у них одни и те же главный функтор и арность, и каждая из их соответствующих компонент сопоставима. Если  $X$  и  $Y$  – неконкретизированные (свободные) переменные, то они сопоставимы, и в этом случае говорят, что они сцеплены.

На рис. 1.1 приведены примеры отождествимых и неотождествимых термов.

Терм1	Терм2	Отождествимы?
иван (X)	иван(человек)	Да: $X=человек$
иван (личность)	иван(человек)	Нет
иван (X,X)	иван(23,23)	Да: $X=23$
иван (X,X)	иван(12,23)	Нет
иван ( , )	иван(12,23)	Да
$F(Y,Z)$	$X$	Да: $X=f(Y,Z)$
$X$	$Z$	Да: $X=Z$

Рис. 1.1

Заметим, что Пролог находит наиболее общий унификатор термов. В последнем примере на рис. 1.1 существует бесконечное число унификаторов:

$$X=1, Z=1; X=2, Z=2 \dots$$

Но Пролог находит наиболее общий –  $X=Z$ .

Возможность отождествления двух термов проверяется с помощью оператора  $=$ .

Ответом за запрос

$$? 3+2=5.$$

будет «нет», так как термы неотождествимы (оператор не вычисляет значения своих аргументов), но попытка доказать:

$$? строка(поз(X))=строка (поз(23)).$$

закончится успехом при  $X=23$ .

Унификация часто используется для доступа к подкомпонентам термов. Так, в приведенном выше примере  $X$  конкретизируется первой компонентой терма  $поз(23)$ , который, в свою очередь, является компонентой терма строки.

### 1.3. РЕКУРСИЯ

Рекурсия является мощным методом программирования. В Прологе она приобретает особую важность, так как в этом языке отсутствуют конструкции циклов, присущие обычным языкам программирования.

Обычная стратегия решения задач состоит в том, чтобы разбить исходную задачу на более мелкие подзадачи, решить их, а затем объединить подзадачи, с тем чтобы получить решение исходной задачи.

Если подзадача есть уменьшенный вариант исходной задачи, то способ ее разбиения и решения идентичен примененному к исходной задаче. Такой процесс называется *рекурсией*.

Для того чтобы описанный метод решения был результативным, он должен в конце концов привести к задаче, решаемой непосредственно. Решить ее позволяют утверждения, называемые граничными условиями.

### Пример 1.3.1

Рассмотрим способы определения  $n$ -го термина в последовательности 1, 1, 2, 6, 24, 120, 720...

Первый вариант описания таков: нулевой терм равен 1, первый терм получается при умножении  $1*1$ , второй терм – при умножении  $1*1*2$ , третий – при умножении  $1*1*2*3$ ,..., в общем случае  $n$ -й терм получается при умножении  $1*1*2*3*...*n$ .

С другой стороны, нулевой терм есть 1, а  $n$ -й терм есть  $(n-1)$ -й терм, умноженный на  $n$ . Это определение является рекурсивным, поскольку разбивает задачу нахождения  $n$ -го термина на задачи нахождения с начала  $(n-1)$ -го термина и умножения его затем на  $n$ .

Ясно, что оба определения эквивалентны, так как задача вычисления третьего термина с помощью рекурсивного определения сводится к задаче вычисления второго термина и умножения его на 3. Задача вычисления второго термина сводится к задаче вычисления нулевого термина и умножения его на 1. Поскольку нулевой терм равен 1, вычислим первый терм – он будет равен 1. Это, в свою очередь, позволяет нам вычислить второй терм – он равен 2, а затем – третий терм, равный 6.

Для обозначения факта, что  $N$ -й член последовательности равен  $V$ , воспользуемся предикатом *посл*( $N, V$ ). Рекурсивное определение выражается следующими утверждениями Пролога:

/\*Граничное условие – нулевой терм равен 1\*/  
*посл*(0,1).

/\*Рекурсивное условие – если  $(N-1)$ -й терм равен  $U$ , то  $N$ -й терм равен  $U*N$ \*/

*посл* ( $N,V$ ):- $M=N-1$ ,*посл*( $M,U$ )= $U*N$ ,  $V = U*N$ .

Ответом на вопрос ?*посл*(3, $Z$ ) будет  $Z=6$ .

## 1.4. ОПИСАНИЕ РАБОТЫ ПРОЛОГА ПО ПОИСКУ РЕШЕНИЯ ЗАДАЧИ

Для того чтобы представить, каким образом Пролог находит ответ, рассмотрим вопрос ?*посл*(3,*Z*) и опишем работу Пролога на двух этапах: разбиения и решения задачи.

### ЭТАП РАЗБИЕНИЯ

**Первое обращение.** Пролог пытается удовлетворить запрос *посл*(3,*Z*), используя первое утверждение процедуры описания последовательности *посл*(0,1). Поскольку первая компонента терма (0) не сопоставляется с первой компонентой запроса (3), попытка заканчивается неудачей. Теперь Пролог пытается использовать второе утверждение, описывающее последовательность. На этот раз заголовок второго утверждения *посл*(*N*,*V*) сопоставляется с запросом *посл*(3,*X*), при этом *N* сопоставляется со значением 3, а *V* связывается с *X*. Пролог переходит к доказательству целей в теле утверждения:

/\*  $N=3$

/\*  $V=X$

$M = 3-1$ , *посл*(*M*,*U*), =  $U*3$ . /\* откладывается

Первое целевое утверждение согласуется при  $M=2$ . Второе целевое утверждение *посл*(2,*U*) приводит ко второму рекурсивному обращению. Пролог пытается согласовать третье целевое утверждение только при условии, что согласовано второе. Поэтому третье целевое утверждение откладывается.

**Второе обращение.** Для того чтобы согласовать *посл*(2,*U*), Пролог пытается сопоставить *посл*(2,*U*) с первым утверждением процедуры *посл*(0,1), но неудачно. поскольку первые компоненты термов не сопоставляются.

Однако удастся сопоставить *посл*(2,*U*) с головой второго утверждения *посл*(*N*<sub>2</sub>,*V*<sub>2</sub>) при *N*<sub>2</sub>, равной 2, и *V*<sub>2</sub>, равной *U*. Теперь Пролог пытается согласовать тело утверждения:

/\*  $N_2=2$

/\*  $V_2=U$

$M_2$  is  $2-1$ , *посл*(*M*<sub>2</sub>,*U*<sub>2</sub>),  $U$  is  $U_2*2$ . /\* откладывается.

Мы использовали то же утверждение, что и в первом обращении, но с меньшим аргументом  $M2$ .

Чтобы отличить второе обращение к утверждению от первого, присвоим переменным индекс 2. В общем случае индекс  $n$  у переменной показывает, что она применяется при  $n$ -м обращении. Переменные при первом обращении записаны без индексов.

Первое из целевых утверждений согласуется при  $M2$ , равной 1. Второе целевое утверждение приводит к третьему обращению, а третье целевое утверждение откладывается.

**Третье обращение.** Чтобы согласовать  $посл(1, U2)$ , Пролог пытается сопоставить его с первым утверждением определения  $посл(0, 1)$ , но неудачно. Однако удастся сопоставить  $посл(1, U2)$  с заголовком второго утверждения, причем  $N3$  получает значение 1, а  $V3$  связывается с  $U2$ . Следовательно, теперь надо согласовать цели:

/\*  $N3=1$

/\*  $V3=U2$

$M3$  is 1-1,  $посл(M3, U3)$ ,  $U2 = U3*1$ . /\* откладывается

Первое целевое утверждение согласуется при  $M3$ , равной 0. Второе целевое утверждение  $посл(0, U3)$  Пролог пытается согласовать, сопоставляя его с первым утверждением определения. На этот раз два термина успешно сопоставляются при  $U3$ , равной 1.

Редукция задачи завершена, и граничные условия позволили решить последнюю задачу. Теперь Пролог возвращается к отложенным целевым утверждениям и пытается согласовать самое последнее из них. Если удастся это сделать, Пролог переходит к согласованию предпоследнего отложенного целевого утверждения и так до тех пор, пока не будет согласовано первое. В представленном примере мы предполагали, что все отложенные целевые утверждения успешно согласуются. Далее (в разделе 1.5) мы покажем, что происходит, если Пролог не может согласовать целевое утверждение.

## ЭТАП РЕШЕНИЯ ЗАДАЧИ

**Согласование *посл(1,U2)*.** Самое последнее из отложенных целевых утверждений – третье целевое утверждение в третьем обращении:  $U2 \text{ is } U3 * 1$ . Поскольку переменная  $U3$  равна 1 в результате согласования второго целевого утверждения,  $U2$  получает значение 1. Таким образом, *посл(1,U2)* согласуется при  $U2$ , равной 1.

**Согласование *посл(2,U)*.** Предпоследнее отложенное целевое утверждение – третье целевое утверждение во втором обращении:  $U = U2 * 2$ . Так как переменная  $U2$  равна 1 в результате согласования второго целевого утверждения,  $U$  получает значение 2. Итак, *посл(2,U)* согласуется при  $U$ , равной 2.

**Согласование *посл(3,X)*** Предыдущее из отложенных целевых утверждений – третье целевое утверждение в первом обращении:  $X = U * 3$ . Поскольку переменная  $U$  равна 2 в результате согласования второго целевого утверждения,  $X$  получает значение 6. Целевое утверждение *посл(3,X)* согласуется при  $X$ , равной 6.

Если отложенных целевых утверждений больше нет, Пролог определяет, при каком значении переменной удовлетворяется запрос:

$$X = 6$$

другие решения (да/нет)? Нет.

На рис. 1.2 показана схема, отражающая полную трассировку запроса. Дуги, исходящие из цели, помечены номером сопоставляемого утверждения и значениями переменных при сопоставлении. Значения переменных, при которых согласуется целевое утверждение, заключены в круглые скобки и записаны на той же строчке, что и соответствующее целевое утверждение. Метки на дугах, входящих в вершины со значениями переменных, являются целевыми утверждениями, которые приводят к этим значениям.

### Пример 1.3.2

*Требуется написать программу для вычисления значения суммы ряда:*

$$F(x, n) = 1 + x + x^2 + x^3 + \dots + x^n$$

*при известных действительном числе  $x$  и натуральном числе  $n$ .*



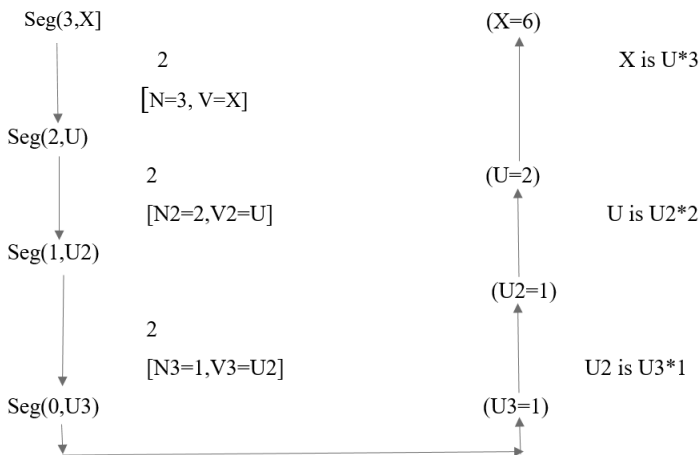


Рис. 1.2. Трассировка запроса  $\text{post}(3, X)$

Поскольку утверждения Пролога определяют некоторое отношение, а не вычисляют значение (как функции в процедурных языках программирования), нам нужно ввести третий аргумент, который обозначал бы сумму  $1 + x + x^2 + x^3 + \dots + x^n$  для заданных  $x$  и  $n$ .

Тогда заголовок правила будет выглядеть следующим образом:

$F(X, N, S)$  (с заглавной буквы в Прологе начинаются имена переменных).

Для того чтобы написать утверждения, мы должны преобразовать приведенное выше определение ряда в рекурсивное определение. Это можно сделать, учитывая, что:

$$F(x, 0) = 1$$

$$F(x, 1) = 1 + x = F(x, 0) * x + 1$$

$$F(x, 2) = 1 + x + x^2 = F(x, 1) * x + 1$$

.....

$$F(x, n) = 1 + x + x^2 + x^3 + \dots + x^n = F(x, n-1) * x + 1$$

Определим ряд рекурсивно:

$$F(x, 0) = 1$$

$$F(x, n) = F(x, n-1) * x + 1$$

Такому определению соответствует следующее утверждение Пролога:

*/\* граничное условие*

$F(X, 0, 1).$

*/\* рекурсивное условие*

$F(X, N, S):-M = N - 1, F(X, M, R), S = R * X + 1.$

## 1.5. МЕХАНИЗМ ВОЗВРАТА В ПРОЛОГЕ

При согласовании целевого утверждения в Прологе используется метод, известный под названием **механизм возврата**. По сути, алгоритм работы движка Пролога основан на методе механизма возврата (бэктрекинга) и носит название «**Поиск в глубину с возвратом**». Опишем метод механизма возврата.

При попытке согласования целевого утверждения Пролог выбирает первое из тех предложений, заголовков которых сопоставим с целевым утверждением. Если удастся согласовать тело утверждения, то целевое утверждение согласовано. Если нет, то Пролог переходит к следующему утверждению, голова которого сопоставима с целевым утверждением, и так далее, до тех пор, пока целевое утверждение не будет согласовано, или не будет доказано, что оно не согласуется с базой данных.

Пример 1.3.

*Меньше* ( $X, Y$ ):- $X < Y$ , *write*( $X$ ), *write*(«меньше, чем»), *write*( $Y$ ).

*Меньше* ( $X, Y$ ):- $Y < X$ , *write*( $Y$ ), *write*(«меньше, чем»), *write*( $X$ ).

Целевое утверждение: *?Меньше*(5,2) сопоставляется с головой первого утверждения при  $X = 5$ ,  $Y = 2$ . Однако не удастся согласовать первый член конъюнкции в теле утверждения  $X < Y$ . Значит, Пролог не может использовать первое утверждение для согласования целевого утверждения *Меньше*(5, 2). Тогда Пролог переходит к следующему предложению, заголовок которого сопоставим с целевым утверждением. В нашем случае это второе утверждение. При значениях переменных  $X = 5$ ,  $Y = 2$  тело утверждения согласуется. Целевое утверждение *Меньше*(5,2) доказано, и Пролог выдает сообщение «2 меньше, чем 5».

Запрос *?Меньше*(2,2) сопоставляется с головой первого утверждения, но тело утверждения согласовать не удастся. Затем происходит сопоставление с головой второго утверждения, но согласовать тело опять-таки оказывается невозможным. Поэтому попытка согласования целевого утверждения *Меньше*(2,2) заканчивается неудачей.

Такой процесс согласования целевого утверждения путем прямого продвижения мы называем **прямой трассировкой**. Даже если целевое утверждение согласовано, с помощью прямой трассировки мы можем попытаться получить другие варианты его доказательства, т. е. вновь согласовать целевое утверждение.

Пролог производит доказательство конъюнкции целевых утверждений слева направо. При этом может встретиться целевое утверждение, согласовать которое не удастся. Если такое случается, то происходит смещение влево до тех пор, пока не будет найдено целевое утверждение, которое может быть вновь согласовано, если не будут исчерпаны все предшествующие целевые утверждения. Если слева нет целевых утверждений, то конъюнкцию целевых утверждений согласовать нельзя. Однако если предшествующее целевое утверждение может быть согласовано вновь, Пролог возобновляет процесс доказательства целевых утверждений слева направо, начиная со следующего справа целевого утверждения. Описанный процесс смещения влево для повторного согласования целевого утверждения и возвращения вправо носит название **механизм возврата – бэктрекинг**.

## **2. РАЗРАБОТКА ПРОСТЕЙШЕЙ БАЗЫ ЗНАНИЙ НА ЯЗЫКЕ ПРОЛОГ**

В этом разделе вы получите первое представление о формально-логической модели представления знаний, реализованной в системе логического программирования, изучите синтаксис и семантику логической программы.

Кроме того, свою первую программу вы создадите и отладите в простейшей среде интерпретатора языка VISUAL PROLOG.

Разработку простейшей программы на языке Пролог мы рассмотрим на примере разработки базы знаний предметной области «Родственные отношения». На примере простейшей предметной области родственных отношений мы научимся описывать факты и правила в формально-логической модели представления знаний.

### **2.1. СИНТАКСИС И СЕМАНТИКА (СМЫСЛ) ЛОГИЧЕСКОЙ ПРОГРАММЫ**

Наиболее сложной моделью представления знаний является формально-логическая, в основе которой лежит формальная логика, а именно – исчисление предикатов. Язык Пролог, с одной стороны – язык программирования для систем искусственного интеллекта, а с другой стороны – язык представления знаний в формально-логической модели. Поэтому его изучение является необходимым условием ознакомления с формально-логической моделью представления знаний, лежащей в основе всех остальных моделей (продукционной модели, семантической сети, фреймов), разработанных для представления знаний. Программу, написанную на языке Пролог, будем в дальнейшем называть логической программой (или базой знаний).

**Логическая программа** представляет собой базу знаний (БЗ), состоящую из предложений.

Каждое **предложение** имеет синтаксис

$$A:- B1, B2, \dots, Bn.$$

Здесь  $A, B1, B2, \dots, Bn$  – предикаты.

Декларативный (логический) смысл предложения:

$$B1 \wedge B2 \wedge \dots \wedge Bn \Rightarrow A,$$

(т. е. в языке Пролог символ «:-» означает импликацию  $\Rightarrow$ , направленную влево, а запятая означает конъюнкцию « $\wedge$ », имеющую смысл логического «И»). Содержательно данное предложение можно прочитать так: «Из истинности  $B1$  и  $B2$  и ...  $Bn$  следует истинность  $A$ ».

**Предикат** – это выражение вида

$$pred(term_1, \dots, term_m)$$

где *pred* – имя предиката (предикатный символ);  $term_1, \dots, term_m$  – термы.

Декларативный смысл (семантика) предиката: предикат задает **отношение** между объектами, задаваемыми термами  $term_1, \dots, term_m$ . Предикат принимает два значения: 1 (истина) либо 0 (ложь). Предикат является истинным, если отношение между объектами имеет место. Предикат является ложным, если отношение между объектами не имеет места.

**Терм** – это константа, переменная или составной терм. Термы указывают на объекты предметной области, связываемые отношениями, задаваемыми предикатами.

**Константа** – число или атом.

**Атом** – цепочка символов, начинается со строчной буквы. Указывает на конкретный объект.

**Переменная** – цепочка символов, начинается с прописной буквы. Переменная указывает на неопределенный объект.

**Составной терм** – выражение вида

$$func(term_1, \dots, term_m),$$

где *func* – имя функции (функциональный символ);  $term_1, \dots, term_m$  – термы. Позволяет реализовать понятие функции.

Полный вид предложения логической программы называется **правилом**:

$$A:- B_1, B_2, \dots, B_n.$$

Правило читается «Если  $B_1$  и  $B_2$  и ... и  $B_n$ , то  $A$ » и состоит из двух частей, разделенных символом «:-», т. е. из заголовка правила « $A$ » и тела правила « $B_1, B_2, \dots, B_n$ ». Если правило состоит только из заголовка, то оно называется фактом. Логический смысл факта – это утверждение, которое всегда истинно (без условия «если»).

Пример программы, состоящей из трех предложений:

```
father("Bill", "John").  
father("Pam", "Bill").  
grandFather(Person, GrandFather):-  
father(Person, Father), father(Father, GrandFather).
```

Данные предложения описывают предметную область родственных отношений. Заметим, что они совершенно независимы друг от друга. Переменные, входящие в одно предложение, никак не связаны с переменными, имеющими те же имена, но входящими в другое предложение. Область действия любой переменной – одно предложение.

Логическая программа обладает как декларативной, так и процедурной семантикой. Причем факты, присутствующие в программе, можно воспринимать только как декларации о предметной области, в которой решается конкретная задача. Это – бесприкословные заявления, описывающие ту или иную ситуацию, а вот правила можно интерпретировать не только как декларации, но и как процедуры программы. Правило в программе на языке Пролог вида

$$A:- B_1, B_2, \dots, B_n$$

можно трактовать двояко, во-первых, как логическое высказывание

$$B_1 \wedge B_2 \wedge \dots \wedge B_n \Rightarrow A,$$

означающее импликацию (истинность конъюнкции  $B_1 \wedge B_2 \wedge \dots \wedge B_n$  влечет за собой истинность заключения  $A$ ). Во-вторых, это правило можно рассматривать как определение процедуры  $A$ , утверждающее, что для выполнения  $A$  надо выполнить  $B_1, B_2, \dots, B_n$ . Замечательным свойством логического программирования является то, что обе семантики (декларативная и процедурная) равноправны и в большой степени независимы, хотя и полностью соответствуют друг другу. Поэтому на

одну и ту же логическую программу человек-программист и компьютер-исполнитель смотрят с различных позиций, наилучшим образом соответствующих стилю мышления каждого из участников процесса программирования.

Семантика записанных выше предложений определяется следующим образом. Имеются три объекта "Bill", "John" и "Pam", связанные отношениями родства. Первое предложение является фактом (безусловной истиной), утверждающим, что человек по имени «John» приходится отцом "Bill". Второе предложение – также факт, который утверждает, что "Bill" – отец "Pam". Третье предложение – правило, оперирующее с переменными. Оно определяет истинность импликации, утверждающей, что если некий GrandFather является отцом человека по имени Father, а Father, в свою очередь, отцом человека по имени Person, то GrandFather является праотцом Person. Хотя при записи предложений кванторы опускаются, предполагается, что переменные в правилах и фактах связаны квантором всеобщности. Истинность конъюнкции в правой части правила влечет за собой истинность заключения для любых значений переменных, для которых устанавливается истинность посылки.

Описание предметной области с использованием правил и фактов является статическим, само по себе оно никакого процесса вычислений не задает, а только определяет базу знаний, в которой хранится информация об объектах и отношениях между ними. Логическая программа начинает выполняться после ввода предполагаемого заключения, которое называется запросом (или целью) к программе и представляет собой условие или конъюнкцию нескольких условий. Например, к описанной выше программе родственных отношений могут быть поставлены следующие запросы:

- Является ли John отцом Sue?
- Кто является отцом Pam?
- Является ли John дедушкой Pam?

Вышеприведенные цели формально записываются следующим образом:

- ?- father("Sue", "John").
- ?- father("Pam", X).
- ?- grandFather("Pam", "John").

В программе должно быть только одно целевое утверждение. Некоторые цели, такие как первая и третья, предполагают односложный ответ "yes" (третья цель) или "no" (первая цель). Такие цели не содержат

переменных и называются базовыми. Другие цели, как, например, вторая, содержат переменные. В результате в случае положительного ответа "yes" на запрос вычисляются значения переменных, содержащихся в запросе. Например, на второй запрос, который не является базовым, будет получено одно решение  $X = \text{"Bill"}$ .

Некоторые запросы могут выдавать множество решений, Например, запрос

?- father(X, Y).

В ответ на этот запрос мы имеем два решения:

$X = \text{"Bill"}, Y = \text{"John"}$ .

$X = \text{"Pam"}, Y = \text{"Bill"}$ .

Таким образом, логическая программа состоит из базы знаний (правила и факты) и цели (целевого утверждения). Выполнение программы заключается в попытке доказать целевое утверждение из теории, сформулированной с помощью высказываний базы знаний.

Имея факты об отцовстве, логично расширить пример родственных отношений высказываниями об отношении материнства. На основе отношений mother и father можно определить отношение parent, например, следующим образом:

parent(Person, Parent):- mother(Person, Parent).

parent(Person, Parent):- father(Person, Parent).

Декларативная семантика этих предложений состоит в следующем. Parent является родителем Person, если Parent является матерью Person (первое предложение), или Parent является родителем Person, если Parent является отцом Person (второе предложение). Так как заголовки двух альтернативных предложений совпадают, то можно воспользоваться более краткой записью с использованием точки с запятой в качестве логической связки « $\vee$ » (дизъюнкция, логическое «ИЛИ»).

parent(Person, Parent):-

mother(Person, Parent);

father(Person, Parent).

Однако рекомендуется использовать точку с запятой как можно реже.



В рассматриваемой программе родственных отношений до сих пор использовались простые типы данных, так называемые **простые домены**. Так, для задания имен использовались данные символьного типа string.

В Прологе заданы следующие стандартные простые домены/

**Integer** – домену integer принадлежат целые числа, находящиеся в пределах от  $-32\,768$  до  $32\,767$ .

**Real** – домену real принадлежат вещественные числа, находящиеся в пределах от  $-1e+307$  до  $+1e+308$ .

**String** – домену string принадлежит строка – последовательность символов между парой двойных кавычек.

**Symbol** – домену symbol принадлежит символическая константа – последовательность символов, начинающаяся со строчной буквы. Строки тоже воспринимаются как элементы типа symbol, но в отличие от строк элементы типа symbol хранятся во внутренней таблице Пролога для быстрого сопоставления.

**Char** – домену char принадлежат символы – 8-битовый ASCII-символ, заключенный в одиночные апострофы.

**File** – домену file принадлежат символические имена файлов. Имя файла должно начинаться со строчной буквы.

Однако люди, участвующие в родственных отношениях, кроме имени могут иметь и другие характеристики, например пол. Для реализации этой возможности в логическом программировании существует возможность использовать **составные домены**, представляющие собой совокупность простых доменов. Составные домены (составные термы, см. определения выше) фактически реализуют понятие функции логики первого порядка и записываются с использованием функторов. Основное преимущество использования функций в логическом программировании в том, что мы можем изменять внутренние аргументы функций без изменения определений большинства предикатов, использующих эти функции.

Рассмотрим составной домен `person(Name,Gender)` с функтором `person` и двумя аргументами, каковыми являются имя человека и его пол. Используя этот составной домен, усовершенствуем построенную ранее программу родственных связей. Однако сначала заметим, что первоначально с использованием исходных фактов мы определяли отношения

father и mother, на основе которых с помощью правил было построено отношение parent. Используя составной домен person(Name,Gender), логичнее поступить наоборот. Исходным отношением положить отношение parent, на основе которого определить все остальные отношения. Ниже представлена усовершенствованная программа родственных отношений.

```
parent(person("Bill", male), person("John", male)).  
parent(person("Pam", female), person("Bill", male)).  
parent(person("Pam", female), person("Jane", female)).  
parent(person("Jane", female), person("Sue", female)).
```

```
father(P, person(Name, male)):-  
    parent(P, person(Name, male)).
```

```
mother(P, person(Name, female)):-  
    parent(P, person(Name, female)).
```

```
grandFather(Person, TheGrandFather):-  
    parent(Person, ParentOfPerson),  
    father(ParentOfPerson, TheGrandFather).
```

```
grandMother(Person, TheGrandMother):-  
    parent(Person, ParentOfPerson),  
    mother(ParentOfPerson, TheGrandMother).
```

Сделаем некоторые замечания по приведенной программе. Во-первых, отметим отличие функций от предикатов: факты для person(Name,Gender) отсутствуют в программе, соответствующая информация вводится через аргументы предиката parent. Во-вторых, отношения grandFather и grandMother в данной программе усовершенствованы: теперь определяются по две бабушки и два дедушки для каждого субъекта. В-третьих, в данном случае для обозначения пола используется тип symbol, для которого символьные данные в отличие от типа string не заключаются в кавычки.

В силу рекурсивности определения функции аргументами составного домена могут быть не только переменные и константы, но и другие функции, например, можно определить функцию «супружеская пара» для использования в предикате isACouple:

```
isACouple(Acouple):-  
    ACouple=couple(person(Husband,male),  
                    person(Wife,female)).
```

Часто процесс согласования Прологом поставленной цели иллюстрируется графически с помощью построения дерева поиска поставленной цели. Рассмотрим построение такого дерева на следующем примере.

Пусть у нас задана следующая база данных:

roditel(a,b). /\*a является родителем b\*/

roditel(a,c).

roditel(c,v).

roditel(c,g).

roditel(b,n).

roditel(g,d).

vnuk(X,Y):- roditel(X,Z), roditel(Z,Y) /\*Y является внуком X, если существует ет такой Z, что X является родителем Z, и Z является родителем Y\*/

Задана цель:

? vnuk(a,Y).

Дерево поиска решения для этой цели выглядит следующим образом (рис. 2.1).

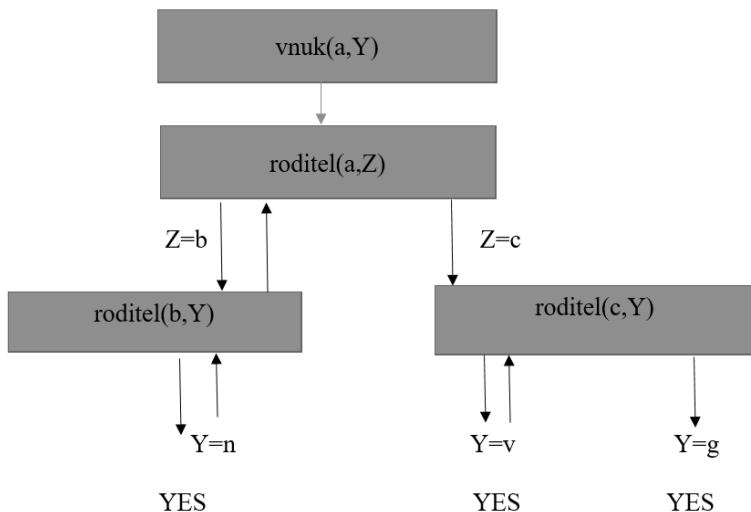


Рис. 2.1

В дереве поиска решений в вершинах (обозначены прямоугольниками) находятся основная цель и промежуточные подцели. Рядом с ребрами-стрелками записываются те унификации, которые осуществляются Прологом при переходе от одной подцели к другой. Дерево поиска решений иллюстрирует алгоритм работы движка Пролога по поиску заданной цели.

В процессе работы Пролог реализует алгоритм унификации. Для этого Пролог использует служебный участок памяти, называемый стеком, а также две служебные переменные  $\Theta$  и  $Y$ . В стек заносятся последовательно те унификации, которые Пролог выполняет в процессе поиска решения. Как только очередное решение найдено, Пролог начинает процедуру бэктрекинга, и из стека удаляются все унификации, выполненные ранее до точки отката.

Изначально содержимого переменная  $\Theta$  не содержит, если в процессе реализации алгоритма унификации находится решение, то оно помещается в эту переменную. Переменная  $Y$  необходима для хранения количества найденных Прологом решений.

Для нашего примера реализация алгоритма унификации выглядит следующим образом.

Шаг 1

*Стек* { vnuk(a,Y) | roditel(a,Z) | (Z,b) }

$\Theta = \{0\}$

$Y = \{0\}$

Шаг 2

*Стек* { vnuk(a,Y) | roditel(a,Z) | (Z,b) | roditel(b,Y) | (Y,n) }

$\Theta = \{n\}$

$Y = \{1\}$

Шаг 3

*Стек* { vnuk(a,Y) | roditel(a,Z) }

$\Theta = \{0\}$

$Y = \{1\}$

Шаг 4

*Стек* { vnuk(a,Y) | roditel(a,Z) | (Z,c) }

$\Theta = \{0\}$

$Y = \{1\}$

Шаг 5

*Стек* { vnuk(a,Y)| roditel(a,Z)|(Z,c)| roditel(c,Y)|(Y,v)}

$\Theta = \{v\}$

$Y = \{2\}$

Шаг 6

*Стек* { vnuk(a,Y)| roditel(a,Z)|(Z,c)}

$\Theta = \{0\}$

$Y = \{2\}$

Шаг 7

*Стек* { vnuk(a,Y)| roditel(a,Z)|(Z,c)| roditel(c,Y)|(Y,g)}

$\Theta = \{g\}$

$Y = \{3\}$

## 2.2. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В процессе выполнения практического задания вы должны составить экспертную систему по анализу родственных связей некоторой произвольно выбранной (самостоятельно) группы людей.

Синтаксис языка Пролог требует, чтобы все идентификаторы констант и предикатов начинались со строчной буквы, а все идентификаторы переменных начинались с прописной буквы. В настоящем учебном пособии для удобства чтения договоримся о другом стиле: идентификаторы, начинающиеся с заглавной или строчной русской буквы, являются константами или предикатами.

План работы:

- 1) построение генеалогического дерева;
- 2) перевод генеалогического дерева в базу фактов;
- 3) составление правил искомых родственных связей в базе знаний;
- 4) примеры запросов к базе знаний и ее ответов.

### 2.2.1. ПОСТРОЕНИЕ ГЕНЕАЛОГИЧЕСКОГО ДЕРЕВА

Перед началом составления базы знаний необходимо составить генеалогическое дерево и изобразить его на бумаге в виде семантической сети (рис. 2.2). Личности, объединяемые генеалогическим деревом, могут быть как реальными (Иван Грозный, Петр I, Николай II, Пушкин А.С. и т. п.), так и виртуальными (родословные гномов и других

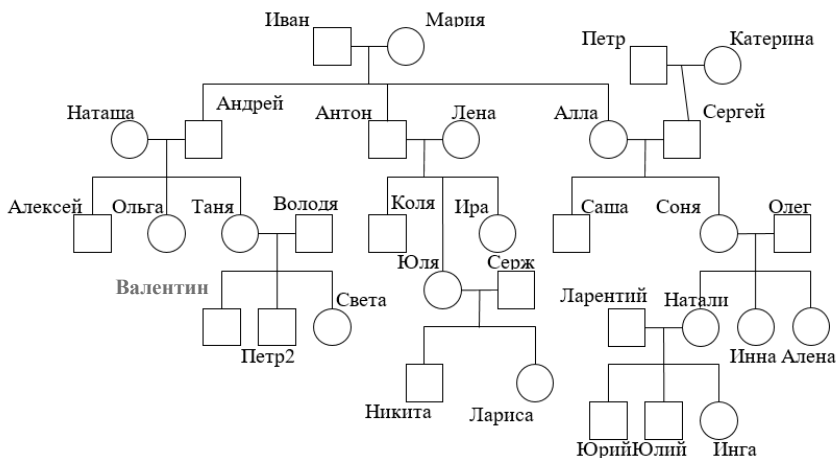


Рис. 2.2

персонажей). Для проверки корректности составления всех правил родственных отношений дерево должно содержать не менее четырех уровней, или, говоря языком предметной области, на дереве должны быть прадедушки (прабабушки) и правнуки (правнучки). Для упрощения задания *не рекомендуется* (но не возбраняется) вводить в генеалогическое дерево личностей, порождающих родственные отношения типа «сводный брат» или «сводная сестра», а также личностей, порождающих циклические связи, являющимися, например, одновременно «супругой» и «двоюродной сестрой». Пример генеалогического дерева приведен на рис. 2.2.

### 2.2.2. ПЕРЕВОД ГЕНЕАЛОГИЧЕСКОГО ДЕРЕВА В БАЗУ ФАКТОВ

При выполнении практического задания вам необходимо для описания базы фактов использовать минимальный набор родственных отношений. Такими отношениями могут быть как вертикальные связи:

Отец (Иван, Андрей). Мать (Мария, Андрей),

Сын (Андрей, Иван). Дочь (Алла, Иван),

так и горизонтальные связи:

Муж (Сергей, Алла). Жена (Алла, Сергей),

Брат (Петр2, Света). Сестра (Света, Петр2),

а также унарные отношения:

Мужчина (Виктор). Женщина (Алла).

Унарные отношения необходимы в тех случаях, когда пол личности невозможно вычислить через существующие отношения. Например, отец всегда является мужчиной, но вот пол того, кому он отец, не определен. Сходная проблема возникает в ситуации с отношениями «брат», «сестра», «сын», «дочь».

Для выполнения задания определите собственные отношения (например, на основе информации о вашей семье):

отец/2. мать/2. мужчина/1.

сын/2. дочь/2. мужчина/1.

сын/3. дочь/3.

супруг/2. женщина/2.

муж/2. сын/2. дочь/2.

родитель/2. женщина/1.

отпрыск/2. мужчина/1.

сестра/2 брат/2. родитель/2.

родители/3. женщина/1.

жена/2. сын/2 дочь/2.

Значками «/1», «/2», «/3» задается арность отношения, или, говоря в терминах предметной области, количество личностей в отношении. Например:

- отношение «мужчина/1» характеризует такой факт, как «Виктор является мужчиной», или на Пролог Мужчина (Виктор);

- отношение «отец/2» характеризует родственное отношение «Иван отец Андрея», или на Пролог Отец (Иван, Андрей);

- отношение «сын/3» характеризует родственное отношение «Андрей сын Ивана и Марии», или на Пролог Сын (Андрей, Иван, Мария);

- отношение «родитель/3» характеризует родственное отношение «Иван и Мария родители Андрея», или на Пролог Родители (Иван, Мария, Андрей).

По обоснованному решению студента минимальный набор родственных отношений может быть расширен дополнительными отношениями в целях полноты описания генеалогической структуры родственных связей.

Для наглядности каждое родственное отношение можно нарисовать стрелочкой (бинарные отношения) или кружочком (унарные отношения) определенного цвета, уникального для каждого типа отношения.

Затем все нарисованные отношения необходимо переписать в виде фактов. Необходимым условием правильности перевода семантической сети в факты является соответствие количества стрелок каждого цвета и количества фактов этого отношения. Каждой стрелке должен соответствовать только один факт.

### 2.2.3. СОСТАВЛЕНИЕ ПРАВИЛ ИСКОМЫХ РОДСТВЕННЫХ СВЯЗЕЙ В БАЗЕ ЗНАНИЙ

Далее вам необходимо написать правила для установления родственных связей (всем студентам), таких как:

- отец, мать, сестра, брат, сын, дочь, муж, жена;
- двоюродная сестра (кузина), двоюродный брат (кузен), дядя, тетя, племянник, племянница;
- дедушка, бабушка, прадедушка, прабабушка, внук, внучка, правнук, правнучка;
- кровные (общий предок), сводные брат и сестра, шурин, деверь, золовка, тесть, теща, зять, невестка и т. п. родственные отношения.

Для каждого родственного отношения первоначально необходимо записать правило на естественном языке (в виде продукции). Например:

**ЕСЛИ** есть такой  $X$  который является отцом  $Z$

**И**  $Z$  является отцом **ИЛИ** матерью  $Y$

**ТОГДА**  $X$  является дедом  $Y$ .

Затем переписать это же правило на языке Пролог:

Дед ( $X, Y$ ):- Отец ( $X, Z$ ), ( Отец ( $Z, Y$ ); Мать ( $Z, Y$ ) ).

В синтаксисе языка Пролог это правило будет выглядеть так:

grandfather ( $X, Y$ ):- father ( $X, Z$ ), ( father ( $Z, Y$ ); mother ( $Z, Y$ ) ).

или можно без перевода на английский язык просто записать латиницей:

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

### 2.2.4. ПРИМЕРЫ ЗАПРОСОВ К БАЗЕ ЗНАНИЙ И ЕЕ ОТВЕТОВ

В этом разделе необходимо привести примеры диалога с созданной экспертной системой. Показать возможные варианты запросов системы и ее ответов. Например:

Запрос: Дед (Олег,  $Y$ )

Ответ 1:  $Y$  = Юрий



Ответ 2:  $Y = \text{Юлий}$

Ответ 3:  $Y = \text{Инга}$

Пример сложного составного запроса:

$\text{Сестра}(X, Y), \text{Сестра}(Y, X).$

## СОДЕРЖАНИЕ ПРАКТИЧЕСКОЙ РАБОТЫ

1. Разработать базу знаний, содержащую сведения о генеалогическом дереве вашей семьи. Разработать правила, отражающие следующую семантику созданной базы данных:

– *Всякий, кто имеет ребенка, – счастлив.*

– *Всякий  $X$ , имеющий ребенка, у которого есть сестра или брат, имеет двух детей.*

– *Определите отношение  $\text{ВНУК}(X, Y)$ , используя отношение  $\text{РОДИТЕЛЬ}$ .*

– *Определите отношение  $\text{ТЕТЯ}(X, Y)$  через отношения  $\text{РОДИТЕЛЬ}$  и  $\text{СЕСТРА}$ .*

2. Оформить отчет.

### *Содержание отчета*

1. Текст задания.

2. Генеалогическое дерево.

3. Перевод генеалогического дерева в базу фактов.

4. Составление правил искомых родственных связей в базе знаний.

5. Примеры запросов к базе знаний и ее ответов.

6. Листинг разработанной базы знаний на языке Пролог.

7. Описание декларативного смысла фактов и предложений разрабатываемой программы.

8. Описание декларативного смысла целей.

### **3. РАЗРАБОТКА ПРОГРАММ НА ПЛАТФОРМЕ VISUAL PROLOG**

В этом разделе вы ознакомитесь с особенностями работы оболочки VISUAL PROLOG версий 4-5.2 для языка Пролог, с принципами программирования и структурой программы в VISUAL PROLOG.

Кроме того, на примере простейших баз данных вы научитесь описывать факты и простейшие отношения между объектами в формально-логической модели представления знаний.

#### **3.1. ОСНОВНЫЕ НАЧАЛЬНЫЕ СВЕДЕНИЯ О PROLOG И VISUAL PROLOG**

1. Программы на Прологе включают в себя фразы двух типов (также известных под названием «предложения»): факты и правила.

Факты соответствуют связям и свойствам, о которых вы знаете, что они всегда истинны.

Правила соответствуют зависимым связям, так как они позволяют Прологу выводить одно утверждение из другого. Заключение правила становится истинным, если доказывается, что истинно заданное множество условий (по правилу Modus Ponens). Применимость каждого правила зависит от доказательства истинности соответствующих условий.

2. Все правила в Прологе имеют две части: заголовок и тело, соединенные символом "*if*". Символ "*if*" можно заменить обозначением ":-". Заголовок содержит заключение правила. Тело – это множество (конъюнкция) условий, которые должны быть истинны, для того чтобы Пролог мог доказать истинность заголовка правила.

Факт – это частный случай правила. Факт – это правило, состоящее из одного заголовка (не содержащее тела).

3. Определив множество фактов и правил (базу знаний), вы можете задавать вопросы (формировать цель): это называется запросом к системе. Вопрос – это частный случай правила (правило без заголовка), содержащий только условия, требующие доказательства. Вопрос может быть конъюнктивным (конъюнкция целей).

4. Для доказательства цели (ответа на запрос) механизм логического вывода ищет в программе правило или факт, заголовок которого сопоставляется (унифицируется) с целью. Пролог всегда ищет решение, начиная с первого факта или правила в программе, и просматривает их до достижения самого последнего. Найдя нужное правило, механизм вывода берет из тела этого правила условия (поочередно, слева направо) и пытается их доказать, ставя новые цели для разрешения. Если все условия из тела правила оказываются истинны (доказаны из программы), то цель считается истинной. Если же не все условия истинны, использование правила ни к чему не приводит. В этом случае система логического вывода пытается найти другое правило для доказательства исходной цели. Если же ей не удастся этого сделать, то поставленная цель не может быть выведена из программы (является ложной).

5. Программа на VISUAL PROLOG имеет следующую общую структуру:

```
domains      /*...объявление доменов... */
predicates   /*...объявление предикатов... */
goal         /*...подцель_1, подцель_2, и т. д.... */
clauses      /*...предложения (факты и правила)... */
```

6. В секции **clauses** размещаются факты и правила – база знаний предметной области.

7. В секции **predicates** объявляются предикаты и типы (домены) аргументов этих предикатов. Имена предикатов должны начинаться с буквы (строчной), за которой следует последовательность латинских букв, цифр и символов подчеркивания (до 250 знаков). В именах предикатов нельзя использовать символы пробела, минуса, звездочки, обратной и прямой черты.

Объявление предиката имеет следующую форму:

```
predicates
    predicateName (argument_type1, argument_type2, ...,
                    argument_typeN)
```

Здесь *argument\_type1*, *argument\_type2*, ..., *argument\_typeN* – либо стандартные домены, либо домены, объявленные в секции **domains**.

Объявление домена аргумента и описание типа аргумента-суть одно и то же.

8. Секция *domains* – это секция, в которой объявляются любые нестандартные домены, используемые для аргументов предикатов. Домены в Прологе являются аналогами типов в других языках. Основными стандартными доменами VISUAL PROLOG являются *char*, *integer*, *real*, *string* и *symbol* (строки типа *string* непременно заключаются в кавычки в отличие от строк *symbol*); более подробно стандартные домены описаны в следующих разделах. Основная форма объявления доменов имеет следующий вид:

```
domains  
argument _type1, ..., argument _typeN = < стандартный домен >  
argument 1, ..., argument N = < составной домен 1 >;  
                                < составной домен 2 >;  
                                <...>;  
                                < составной домен N >.
```

9. В секции *goal* задается внутренняя цель программы, что позволяет программе запускаться независимо от среды разработки. Если внутренняя цель включена в программу, то Пролог выполняет поиск только первого решения, и связываемые с переменными значения не выводятся на экран. Если внутренняя цель не используется, то в процессе работы вы будете вводить в диалоговом окне внешнюю цель. При использовании внешней цели (без использования секции *goal*) Пролог ищет все решения и выводит в диалоговое окно все значения, связываемые с переменными, присутствующими в цели.

**Замечание.** В Visual Prolog 5.2 допускается только вариант внешней цели. Причем, если вы хотите, чтобы цель допускала несколько решений (была недетерминированной), предикаты в цели должны быть объявлены с дополнительным определением *nondeterm*, например,

```
nondeterm parent (symbol,symbol)
```

10. Арность (размерность) предиката – это число принимаемых им аргументов. Два предиката с одним именем могут иметь различную арность. Предикаты с одним именем, но с различными версиями арности должны собираться вместе и в секции *predicates*, и в секции *clauses*; при этом предикаты с различными арностями (но с одним именем) рассматриваются как абсолютно разные.

11. Правила имеют форму:

*ЗАГОЛОВОК* : - <Подцель1>, <Подцель2>, ..., <ПодцельN>.

Для разрешения правила интерпретатор должен разрешить все его подцели, создав при этом соответствующее множество связанных переменных. Если же одна из подцелей ложна, Пролог возвратится назад и просмотрит альтернативные решения предыдущих подцелей, а затем вновь пойдет вперед, но с другими значениями переменных. Такой процесс называется «поиск с возвратом» – бэктрекинг.

13. Менее часто используемыми секциями программ на Турбо-Прологе являются секции *database* и *constanes*, а также различные глобальные (*global*) секции (секция *database* будет нами использоваться далее). В *Visual Prolog* эта секция может также называться *facts*.

14. Для работы в системе VISUAL PROLOG 5.2 в режиме обработки цели (без создания проекта) откройте исходный файл (с расширением *pro*), после чего нажмите сочетание клавиш **Ctrl+G**. Заданная цель будет выполнена.

## 3.2. ОБЪЯВЛЕНИЯ И ОПРЕДЕЛЕНИЯ В VISUAL PROLOG

В Прологе, когда необходимо использовать какой-либо предикат или объединенный домен, мы просто делаем это без каких-либо предварительных указаний интерпретатору Пролога [12]. Но в VISUAL PROLOG, прежде чем написать код в разделе *CLAUSES*, мы должны сначала объявить о существовании такого предиката. Такое же объявление должно быть сделано и для любых доменов. Такие объявления позволяют интерпретатору VISUAL PROLOG выявлять дополнительные ошибки в коде программы на этапе компиляции.

### 3.2.1. КЛЮЧЕВЫЕ СЛОВА

Программа, написанная на VISUAL PROLOG, состоит из нескольких разделов, каждый из которых имеет свое название, заданное определенным ключевым словом. Ключевых слов, обозначающих окончание раздела, нет. Необходимо знать следующий перечень ключевых слов.

## **Implement и end implement**

VISUAL PROLOG обрабатывает код, записанный между этими ключевыми словами, как код, принадлежащий одному классу. Имя класса задается после ключевого слова **implement**. Только эти два ключевых слова используются всегда в виде пары.

## **Open**

Это ключевое слово используется для расширения области видимости класса. Оно может применяться только после того как использовалось ключевое слово **Implement**.

## **Constatns**

Это ключевое слово используется для обозначения секции программного кода, в которой определяются константы, многократно используемые в программном коде. Например, если строка «PDC Prolog» будет использоваться в нескольких местах программного кода, то можно определить в секции **Constatns** эту константу-строку следующим образом:

```
Constatns  
pdc = «PDC Prolog».
```

Обратите внимание, что определение постоянной заканчивается точкой «.». В отличие от переменных Пролога имя константы должна быть словом, начинающимся со строчной (маленькой) буквы.

## **domains**

Это ключевое слово используется для обозначения секции, в которой определяются домены, используемые в программном коде (аналог определения типов в других языках программирования). При этом используются четыре характерных формата:

```
name=f /* стандартный домен */  
list=spisok* /* списковый домен */  
compound=f1(d1,d2,...dn);f2(p1,p2,...pl) /* домен составных объектов */  
file=fname1;fname2 /* файловый домен */
```

## **class facts**

Это ключевое слово используется для обозначения секции, в которой объявляются факты, используемые в программном коде. Каждому

факту присваивается уникальное имя, объявляются типы (домены) аргументов, используемые в фактах.

### **class predicates**

Это ключевое слово используется для обозначения секции, в которой объявляются предикаты, содержащиеся в дальнейшем в секции **clauses** программного кода. Предикат объявляется при помощи его имени и доменов его аргументов:

Predname (dom1,dom2,...domm).

В объявлении предикатов, если типы их аргументов являются стандартными (integer, real,string, symbol, char), они записываются в описании предиката в явном виде с использованием соответствующих ключевых слов.

### **clauses**

Это ключевое слово обозначает секцию программного кода, содержащую фактические значения ранее объявленных предикатов – предложения программы. Предложение – это факт или правило, соответствующее одному из объявленных предикатов. Содержимое этой секции совпадает с содержимым аналогичной секции традиционной программы на Прологе. Синтаксис фактических определений предикатов в этой секции полностью соответствует синтаксису, заявленному для этих предикатов в разделе **class predicates**.

### **goal**

В этой секции определяется цель – главная точка входа в программу, написанную на VISUAL PROLOG. Ниже приводится детализированное объяснение понятия цели.

## **3.2.2. ПОНЯТИЕ ЦЕЛИ В VISUAL PROLOG**

В традиционном Прологе, если конкретный предикат определен в программном коде, интерпретатору Пролога в диалоговом режиме может быть дана команда «начать выполнение программы» (поиск решения) с этого конкретного предиката. Но другая ситуация складывается в VISUAL PROLOG: так как компилятор VISUAL PROLOG должен выполнять поиск решения наиболее эффективным способом. Но, в отличие от интерпретаторов, компилятор не может одновременно производить компиляцию программы и выполнять поиск решения. Поэтому поиск решения может быть начат только после компиляции программного

кода, и при этом компилятор должен заранее знать точный предикат, с которого будет начинаться выполнение программного кода. Поэтому в разделе goal записывается эта отправная точка – предикат, с которой начнется поиск цели. Далее программный код компилируется, запоминается. Скомпилированная программа затем работает самостоятельно, без компилятора VISUAL PROLOG или IDE.

### **3.2.3. ФАЙЛОВАЯ СТРУКТУРА ПРОГРАММЫ, НАПИСАННОЙ В СРЕДЕ VISUAL PROLOG**

Если программа, написанная в среде VISUAL PROLOG, имеет большой объем, то хранить ее целиком в одном файле нерационально, поскольку, во-первых, эта программа будет трудночитаема и, во-вторых, это может послужить причиной допущения ошибок при создании программного кода. VISUAL PROLOG позволяет с помощью средств IDE делить программный код на части и хранить каждую часть в отдельном файле. Если поступать таким образом, то к отдельным фрагментам программного кода, используемым в нескольких частях программы, можно получать доступ через файлы. Например, если в программном коде есть домен, используемый в нескольких фрагментах программного кода, записанных в разных файлах, то объявление этого домена осуществляется в отдельном файле, а затем к нему организуется доступ из файлов программного кода, в которых он должен быть использован.

В настоящем учебном пособии для упрощения работы мы будем использовать только один файл для разработки программного кода.

### **3.2.4. ПРОБЛЕМЫ ДОСТУПА К ПРОГРАММАМ БОЛЬШОГО ОБЪЕМА**

В VISUAL PROLOG программные коды большого объема обычно разделяются на отдельные части, при этом в каждой части определяется отдельный класс. В объектно-ориентированных языках программирования класс представляет собой завершенный (упакованный) код, в котором связанные между собой данные хранятся совместно. Фактически, понятие класса в VISUAL PROLOG является синонимом понятия модуля в Турбо-Прологе.

Во время выполнения программы, написанной на VISUAL PROLOG, может возникнуть ситуация, когда необходимо вызвать предикат, определенный в другом классе (файле), или должны стать доступны домены из другого файла.



VISUAL PROLOG позволяет делать такие ссылки-вызовы с использованием специальной концепции, носящей название «Объемный доступ». Рассмотрим реализацию этой концепции на следующем примере.

Пусть предикат `pred1` определен в классе `class1`. Нам необходимо вызвать этот предикат в теле другого предиката `pred2`, определенного в другом файле (`class2`). Далее показано, как это реализуется в программном коде VISUAL PROLOG:

```
clauses
  pred3:-
    ...
    !.

  pred2:-
    class1::pred1,
    pred3,
    ...
```

В этом примере видно, что в теле предиката `pred2` вызываются два предиката `pred1` и `pred3`. Так как `pred1` определен в другом файле, наименование класса `class1` со знаком `::` предшествует имени `pred1`. Предикат `pred3` определен в файле (классе), в котором находится `pred2`. Поэтому наименования класса перед именем `pred3` не требуется.

Таким образом, область видимости предикатов класса ограничивается классом, внутри которого определены эти предикаты. Но область видимости предикатов класса может быть расширена путем использования ключевого слова `open`. Это ключевое слово информирует компилятор о том, что предикаты (а также константы и домены), определенные в заданном классе, доступны и в других классах (файлах) программы.

```
open class1
...

clauses
  pred3:-
    ...
    !.

  pred2:-
    pred1,
    pred3,
    ...
```

В заключение добавим, что VISUAL PROLOG является объектно-ориентированным языком, поэтому (в соответствии с концепцией таких языков программирования) весь программный код любой программы в соответствии с необходимостью разбивается на части, и каждая часть помещается в отдельный класс. Это происходит по умолчанию.

### **3.3. СТАНДАРТНЫЕ ПРЕДИКАТЫ ВВОДА-ВЫВОДА В VISUAL PROLOG**

Для ввода данных и вывода результатов работы на экран вам могут понадобиться стандартные предикаты ввода-вывода Пролога [11]. Опишем наиболее часто используемые из них.

Visual Пролог включает несколько стандартных предикатов для чтения (ввода) данных: `read`, `readln`, `readreal`, `readchar`. Опишем предикат `readln`.

Формат предиката: `readln (Строка)`. Аргумент *строка* имеет тип `string`.

Этот предикат читает строку с текущего устройства ввода и связывает ее с заданной переменной. В качестве конца строки используется символ «Возврат каретки».

Предикатов вывода в VISUAL PROLOG также несколько. Рассмотрим наиболее часто используемый предикат `write`.

Формат предиката: `write(A1,A2,...)`, где `A1`, `A2`,... – константы или переменные.

Этот предикат записывает заданные значения на текущее устройство вывода.

### **3.4. ПРИМЕР ЗАВЕРШЕННОЙ ПРОГРАММЫ НА VISUAL PROLOG**

Рассмотрим пример, который уже был продемонстрирован в разделе 2.1 «Родственные отношения». В приведенном ниже примере весь код написан в среде VISUAL PROLOG IDE. В этом примере несколько больше файлов, чем мы создали в среде VISUAL PROLOG IDE, но эти дополнительные файлы сгенерированы автоматически самой средой IDE. В следующем разделе мы рассмотрим пошаговую инструкцию, которая позволяет разрабатывать такие программные коды в среде VISUAL PROLOG IDE.

Implement main

domains

gender = female; male.

Class facts – familyDB

person : (string Name, gender Gender).

Parent : (string Person, string Parent).

Class predicates

father : (string Person, string Father) nondeterm anyflow.

Clauses

father(Person, Father) :-  
parent(Person, Father),  
person(Father, male).

Class predicates

43randfather : (string Person [out], string GrandFather [out]) nondeterm.

Clauses

43randfather(Person, GrandFather) :-  
parent(Person, Parent),  
father(Parent, GrandFather).

Class predicates

ancestor : (string Person, string Ancestor [out]) nondeterm.

Clauses

ancestor(Person, Ancestor) :-  
parent(Person, Ancestor).  
Ancestor(Person, Ancestor) :-  
parent(Person, P1),  
ancestor(P1, Ancestor).

Class predicates

reconsult : (string FileName).

Clauses

reconsult(FileName) :-  
retractFactDB(familyDB),  
file::consult(FileName, familyDB).

Clauses

```
run() :-  
    console::init(),  
    stdio::write("Load data\n"),  
    reconsult(@"..\fa.txt"),  
    stdio::write("\nfather test\n"),  
    father(X, Y),  
    stdio::writef("% is the father of %\n", Y, X),  
    fail.
```

```
Run() :-  
    stdio::write("\ngrandFather test\n"),  
    44randfather(X, Y),  
    stdio::writef("% is the grandfather of %\n", Y, X),  
    fail.
```

```
Run() :-  
    stdio::write("\nancestor of Pam test\n"),  
    X = "Pam",  
    ancestor(X, Y),  
    stdio::writef("% is the ancestor of %\n", Y, X),  
    fail.
```

```
Run() :-  
    stdio::write("End of test\n").
```

end implement main

```
goal  
mainExe::run(main::run).
```

### 3.5. СОЗДАНИЕ КОНСОЛЬНОГО ПРОЕКТА В СРЕДЕ VISUAL PROLOG IDE [12]

**Шаг 1а.** После запуска VISUAL PROLOG IDE необходимо нажать на пункт меню Project-New.

**Шаг 1б.** Вам откроется диалоговое окно (рис. 3.1). Введите необходимую информацию. Убедитесь, что вы ввели реквизиты будущего проекта Console и NOT GUI.

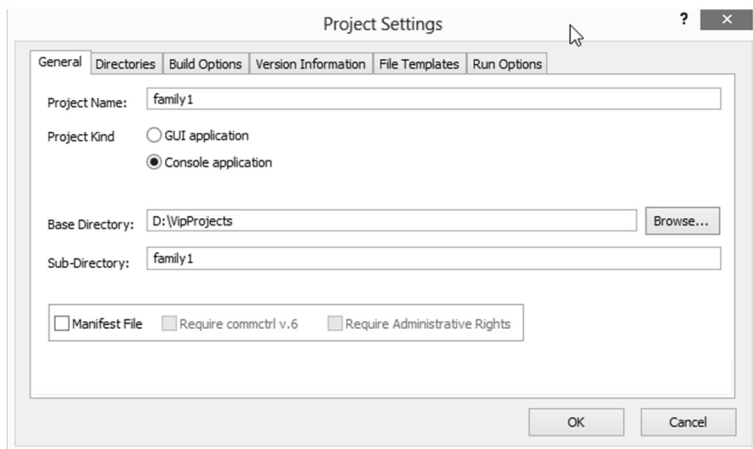


Рис. 3.1

## Шаг 2. Создание пустого проекта

**2а.** После того как вы создали пустой проект, IDE высветит на экране следующее окно проекта. В нем необходимо создать основные файлы исходного кода для проекта.

**2б.** Используйте пункт меню Build-Build для того, чтобы откомпилировать и создать пустой проект (исполняемый файл). Этот файл ничего не делает.

При создании проекта обратите внимание на сообщения, которые динамически отображаются в окне сообщений в среде IDE. Вы заметите, что IDE автоматически вставит в ваш проект все необходимые для работы вашей программы базовые классы VISUAL PROLOG, обеспечивающие в дальнейшем работоспособность вашей программы.

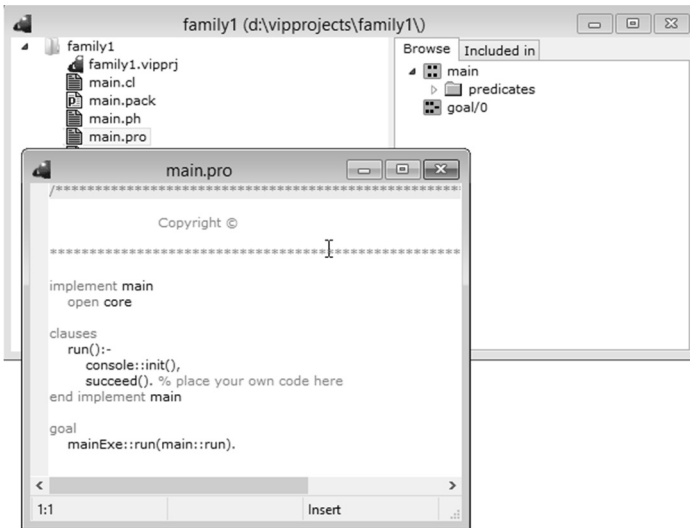
## Шаг 3. Заполнение main.pro фактическим кодом

**3а.** Код, вставленный по умолчанию IDE, ничего полезного не делает. Вам нужно удалить этот код и скопировать фактический код main.pro в это окно (рис. 3.2).

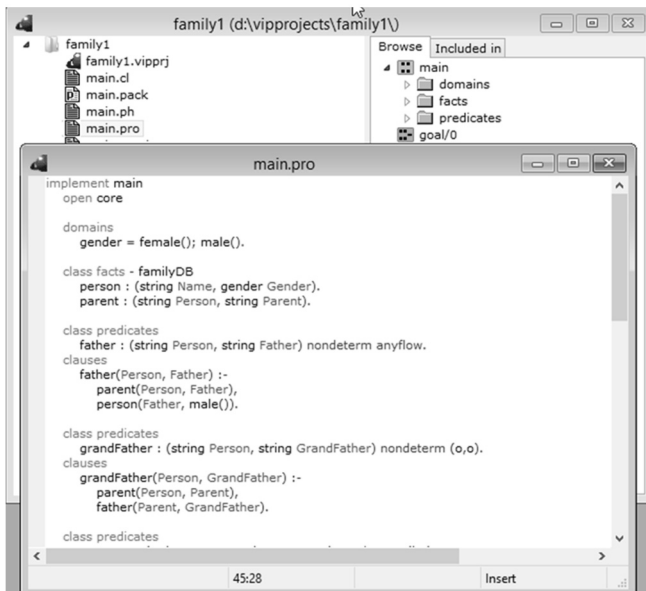
**3б.** После выполнения операций копирования и вставки main.pro окно будет выглядеть так, как показано на рис. 3.3.

## Шаг 4. Перестроение кода

Повторно вызовите команду меню Build. Это сообщит IDE, что исходный код изменился, IDE при этом перекомпилирует программу. Если вы будете использовать команду Build all, то все модули будут перекомпилированы.



*Puc. 3.2*



*Puc. 3.3*

В процессе сборки среда не только выполняет компиляцию, но также и определяет, нужны ли в проекте какие-либо еще модули стандартного интерфейса VISUAL PROLOG(PFC). Если такие модули необходимы, то IDE вставляет их в проект и выполняет повторную компиляцию. Эту ситуацию можно отследить по служебным сообщениям (рис. 3.4).



Рис. 3.4

### Шаг 5. Выполнение программы

Для того чтобы проверить работу только что скомпилированного приложения, можно запустить программу с помощью команды Build-Run. Однако в нашем примере это приведет к ошибке. Причина заключается в том, что наша программа пытается прочитать файл fa.txt, а его нет. Поэтому его необходимо создать с помощью какого-либо текстового редактора и разместить в том же каталоге, где в настоящее время находится наш исполняемый файл. Как правило, исполняемый файл находится в подкаталоге EXE из основного каталога проекта.

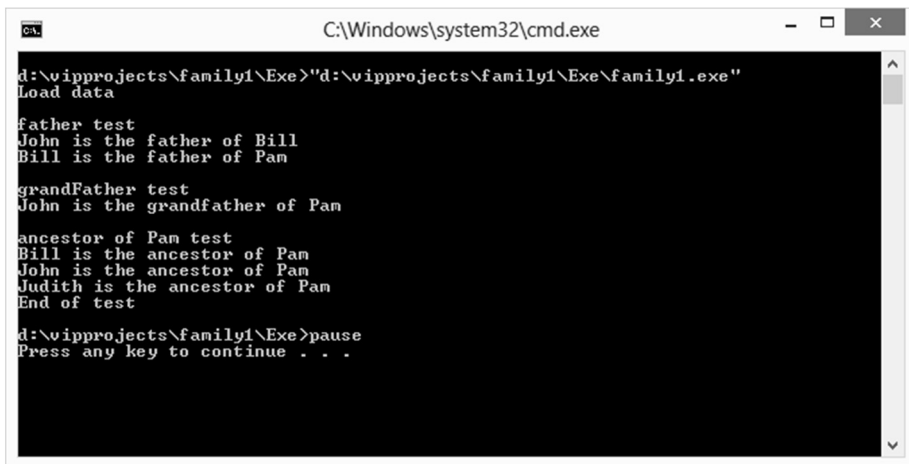
Непосредственно в IDE такой файл создать нельзя, но созданный в текстовом редакторе файл можно сохранить с новым именем (fa.txt) (меню FILE-save as), а затем редактировать этот файл в IDE. После того как файл сохранен, его можно добавить в проект, выбрав пункт меню File-Add.

Содержимое файла fa.txt может быть следующим:

```
clauses
person("Judith",female).
person("Bill",male).
person("John",male).
person("Pam",female).
parent("John","Judith").
parent("Bill","John").
```

Не забывайте, что синтаксис, используемый в файле fa.txt, должен быть совместим с определениями доменов программы!

Теперь, когда вы с помощью пункта меню Build-Run запустите на выполнение программу, вы увидите результат, показанный на рис. 3.5.



```
C:\Windows\system32\cmd.exe

d:\vipprojects\family1\Exe>"d:\vipprojects\family1\Exe\family1.exe"
Load data
father test
John is the father of Bill
Bill is the father of Pam
grandFather test
John is the grandfather of Pam
ancestor of Pam test
Bill is the ancestor of Pam
John is the ancestor of Pam
Judith is the ancestor of Pam
End of test
d:\vipprojects\family1\Exe>pause
Press any key to continue . . .
```

*Рис. 3.5*

Механизмы, которые использует VISUAL PROLOG, являются стандартными (откат и рекурсия).



## 4. ВНУТРЕННЯЯ БАЗА ДАННЫХ VISUAL PROLOG

### 4.1. СПОСОБЫ СОЗДАНИЯ И ОБРАБОТКИ ВНУТРЕННЕЙ БАЗЫ ДАННЫХ В VISUAL PROLOG

Внутренняя база данных состоит из фактов, которые можно непосредственно добавлять в программу на VISUAL PROLOG во время ее исполнения и удалять из нее. Предикаты, описывающие внутреннюю базу данных, объявляются в секции *database* [2].

Опишем, как объявлять секции *database* и как можно изменить содержание вашей внутренней базы данных в процессе выполнения Пролог-программы.

Сразу скажем, что предикаты, описанные в секции *database*, можно использовать таким же образом, как используются предикаты, описанные в секции *predicates*.

Для добавления новых фактов в базу данных в VISUAL PROLOG используются предикаты *asserta* и *assertz*, а предикаты *retract* и *retractall* служат для удаления существующих фактов. Вы можете изменить содержание вашей базы данных, сначала удалив факт, а потом вставив новую версию этого факта (или совершенно другой факт). Предикат *consult* считывает факты из файла и добавляет их к внутренней базе данных, а предикат *save* сохраняет содержимое внутренней секции *database* в файле.

VISUAL PROLOG интерпретирует факты, принадлежащие к базе данных, таким же образом, как и обычные предикаты. Факты предикатов внутренней базы данных хранятся в таблице, их можно легко изменять, тогда как обычные предикаты для достижения максимальной скорости компилируются в двоичный код.

### *Объявление внутренней базы данных*

Для того чтобы воспользоваться внутренней базой данных, необходимо объявить ее в секции *database*. Ключевое слово *database* определяет начало последовательности объявлений предикатов, описывающих внутреннюю базу данных. Секция *database* может выглядеть, например, как в следующем примере:

*domains*

*name, address=string*

*age=integer*

*gender=male; female*

*database*

*person(name, address, age, gender)*

*predicates*

*male(name, address, age)*

*female(name, age, gender)*

*child(name, age, gender)*

*clauses*

*male(Name, Address, Age) :- person(Name, Address, Age, male) ...*

В этом примере вы можете использовать предикат *person* таким же образом, как используются другие предикаты (*male, female, child*). Единственное отличие состоит в том, что можно добавлять и удалять факты для предиката *person* во время работы программы.

Важно отметить следующее:

1) вы можете добавлять в базу данных только факты, но не правила;

2) факты базы данных не могут содержать свободных переменных.

Допускается наличие нескольких секций *database* в программе, но для этого нужно явно указать имя каждой секции *database*. Это требуется потому, что предикаты *consult* и *save* могут загружать и сохранять определенную секцию *database* по имени.

*database – mydatabase*

*myFirstRelation(integer)*

*mySecondRelation(real, string)*

*myThirdRelation(string)*

*goal*

```
consult("example.dba", mydatabase),  
retract(myFirstRelation(1)),  
asserta(myFirstRelation(0)),  
save("example.dba", mydatabase).
```

Такое описание создает базу данных с именем *mydatabase*. Если вы не дадите имени внутренней базе данных, то по умолчанию ей присваивается стандартное имя *dbasedom*.

Важно понимать, что имена предикатов базы данных в модуле уникальны. В двух различных секциях *database* нельзя использовать одинаковые имена предикатов.

Для того чтобы ввести в программу факты, существует три основных способа:

- включение фактов в состав вашей секции *clauses*;
- во время выполнения программы с помощью предикатов *asserta* и *assertz*;
- загрузка файла, содержащего факты базы данных с использованием предиката *consult*.

Все стандартные предикаты VISUAL PROLOG (*asserta*, *assertz*, *retract*, *retractall*, *consult* и *save*) могут иметь один или два аргумента. Необязательный второй аргумент представляет собой имя внутренней базы данных. Опишем эти предикаты. Обозначение «/1» и «/2» после каждого имени предиката указывает необходимое число аргументов для данной версии предиката.

Предикат *asserta* вставляет новый факт в базу данных перед имеющимися фактами для данного предиката, предикат *assertz* вставляет факты после данного предиката.

Добавление фактов имеет следующий формат:

```
asserta(<факт>)  
asserta(<факт>, databaseName)  
assertz(<факт>)  
assertz(<факт>, databaseName)
```

Поскольку имена предикатов базы данных уникальны внутри программы или модуля, для предикатов *asserta* и *assertz* всегда известно, куда нужно добавлять факт. Однако для того чтобы обеспечить работу с нужной базой данных, в целях проверки типа можно использовать необязательный второй аргумент.

Первый предикат следующего примера вставит факт о *Suzanne*, описанный предикатом *person*, после всех фактов *person*, хранящихся в текущий момент в памяти. Второй предикат вставит факт о *Michael* перед всеми имеющимися фактами предиката *person*, третий – вставит факт о *John* после всех других фактов *likes* в базе данных перед всеми другими фактами *likes*.

```
assertz (person("Suzanne", "New Haven", 35, female))
asserta (person("Michael", "New York", 26, male))
assertz (likes("John", "money"), likesDatabase)
asserta (likes("Susanne", "hard work"), likesDatabase)
```

После вызова этих предикатов база данных в памяти с произвольным обращением будет выглядеть так, как будто вы начали работу со следующими фактами:

```
/*Внутренняя база данных – dbasedom*/
person("Michael", "New York", 26, male).
/*... другие факты person ... */
person("Suzanne", "New Haven", 35, female).
/* Внутренняя база данных – likesDatabase*/
likes("John", "money").
/*... другие факты likes ... */
likes("Susanne", "hard work").
```

Предикат *retract(<факт>)* удаляет первый факт во внутренней базе данных, который совпадает с фактом *<факт>*. Этот предикат является недетерминированным (возвращает не одно решение). При поиске с возвратом предикат *retract* возвращает альтернативные решения и удаляет все совпадающие факты до тех пор, пока они имеются. В случае базы данных с именем *databaseName* предикат *retract* имеет следующий формат:

```
retract(<факт>, databaseName).
```

Предположим, в вашей программе имеются следующие секции *database*:

```
database
person(string, string, integer)
database – likesDatabase
```

```

likes(sting, string)
dislikes(string, string)
clauses
/*база данных dbasedom*/
asserta(person("Fred", "Capitola", 35, male)).
assert(person("Fred", "Omaha", 37, male)).
assert(person("Michael", "Brooklin", 26, male)).
/*база данных – likesDatabase*/
assert(likes("John", "money")).
assert(likes("Jane", "money")).
assert(likes("Chris", "chocolate")).
assert(likes("John", "broccoli")).
dislikes("Fred", "broccoli").
dislikes("Michael", "beer").

```

Для программы с такими секциями *database* можно последовательно задать следующие цели:

```

retract(person("Fred", _, _, _))
retract(likes(_, "broccoli"))
retract(likes(_, "money"), likesDatabase)
retract(person("Fred", _, _, _), likesDatabase)

```

Первая цель удалит первый факт *person* о *Fred* из базы данных *dbasedom* (имя которой задается по умолчанию). С помощью второй цели из базы данных *likesDatabase* будет удален первый факт, унифицируемый с *likes(X, "broccoli")*. При этом Турбо-Пролог знает, из какой базы производить удаление, поскольку имена предикатов базы данных уникальны: предикат *person* находится только в неименованной базе данных, а предикат *likes* только в базе *likesDatabase*.

Третья и четвертая цели показывают, как вы можете использовать для проверки типа второй аргумент. Третья цель успешно реализуется, удаляя первый факт, унифицируемый с *likes(X, "money")* из *likesDatabase*, а четвертая – выдаст ошибку, потому что нет (и не может быть) факта *person* в базе данных *likesDatabase*. Сообщение об ошибке выглядит следующим образом:

*506 Type Error: The functor does belong to the domain* (функтор не относится к данному домену).

Следующая цель иллюстрирует, как вы можете получить и вывести на печать все значения базы данных с помощью недетерминированного предиката *retract*:

```
goal
  retract(person(Name, _, Age, _)),
  write(Name, " ", Age),
  fail.
```

С помощью предиката *retract* вы можете также удалить все факты из заданной секции базы данных. Если при вызове *retract* в качестве первого аргумента указана свободная переменная, то для определения того, из какой секции *database* нужно произвести удаление, будет анализироваться второй аргумент. Например:

```
goal
  retract(X, mydatabase), write(X), fail.
```

Предикат *retractall(<факт>)* удаляет из внутренней базы данных все факты, унифицируемые с термом *<факт>*. Этот предикат всегда завершается успешно и является детерминированным. Из *retractall* выходные значения переменных получить нельзя. Это означает, что нужно использовать анонимные переменные, обозначаемые символом подчеркивания «*\_*». Для задания имени базы данных в предикат *retractall* вводится второй аргумент:

```
retractall(<факт>, databaseName).
```

Так же, как и в случае предикатов *assert* и *retract*, для проверки типа можно использовать второй аргумент. Если при вызове предиката *retractall* в качестве первого аргумента используется символ подчеркивания, то из указанной секции *database* можно удалить все факты.

Следующая цель удаляет все факты о мужчинах из базы данных с фактами *person*:

```
retractall(person(_, _, _ male)).
```

Следующая цель удаляет все факты из базы *mydatabase*:

```
retractall(_, mydatabase).
```

Предикат *consult* считывает из файла *fileName* факты, описанные в секции *database*, и вставляет их в конец соответствующей секции

внутренней базы данных (аналогично тому, как предикат *assertz* включает факты). Предикат *consult* имеет один или два аргумента:

```
consult(fileName)  
consult(fileName, databaseName)
```

Однако в отличие от *assertz*, если вы вызовете *consult* только с одним аргументом (без имени базы данных), то в файле должны находиться только факты из указанной базы данных. Если файл содержит что-нибудь кроме фактов указанной базы, то предикат *consult*, когда он дойдет до этого, возвратит ошибку.

Обратите внимание, что предикат *consult* считывает по одному факту. Если файл содержит десять фактов, а в седьмом факте имеется какая-нибудь синтаксическая ошибка, *consult* занесет шесть первых фактов в базу данных, после чего система выдаст сообщение об ошибке.

Отметим, что предикат *consult* может считывать файлы только в том формате, который создает предикат *save* (для включения фактов с максимально возможной скоростью). Файлы не должны содержать:

- прописных символов;
- пробелов, за исключением тех, которые содержатся внутри;
- строк в двойных кавычках;
- комментариев;
- пустых строк;
- символов без двойных кавычек.

При создании или изменении файла с фактами в редакторе, а не программно, нужно соблюдать аккуратность.

Предикат *save* сохраняет факты из указанной базы данных в файле. Этот предикат имеет один или два аргумента.

```
save(fileName)  
save(fileName, databaseName)
```

При вызове предиката *save* только с одним аргументом (без имени базы данных) в файле *fileName* будут сохранены факты из базы данных *databaseName*, используемой по умолчанию.

При вызове предиката *save* с двумя аргументами (имя файла и имя базы данных) в указанном файле будут сохранены факты из секции *databaseName* внутренней базы данных.

Итак, перед выполнением вами практического задания, предлагаемого в настоящем разделе, суммируем полученные сведения.

1. Внутренняя база VISUAL PROLOG состоит из фактов вашей программы, сгруппированных в секции *database*. Предикаты, определяемые пользователем и используемые в этих группах фактов, можно объявлять с помощью ключевого слова *database*.

2. Секциям *database* можно давать имена (с помощью которых создается соответствующий внутренний домен). По умолчанию доменом для неименованной секции *database* является домен *dbasedom*. В программе может присутствовать несколько секций *database*, но каждая из них при этом должна иметь уникальное имя. Заданный предикат базы данных можно описать только в одной секции *database*.

3. С помощью стандартных предикатов *asserta*, *assertz* и *consult* можно добавлять факты к внутренней базе данных во время работы программы. Можно также с помощью предикатов *retract* и *retractall* удалять эти факты во время работы программы.

4. Предикат *save* сохраняет факты из секции *database* в файле (в определенном формате). С помощью редактора можно создавать или редактировать такой файл фактов, а затем можно вносить факты из файла в программу с помощью предиката *consult*.

5. Вы можете обращаться к предикатам базы данных в программе таким же образом, как и ко всем другим предикатам.

6. При использовании внутренних доменов, сгенерированных для секции *database*, можно работать с фактами, как с терминами.

## 4.2. СПОСОБЫ ОРГАНИЗАЦИИ ЦИКЛОВ В VISUAL PROLOG

В предлагаемых к выполнению заданиях встречаются цели, достижение которых Прологом возможно только тогда, когда в правилах базы данных содержатся утверждения, являющиеся аналогами циклов в процедурных языках программирования. Рассмотрим, каким образом реализуются циклы в Прологе [3].

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

### **Первый способ.** Использование рекурсии

Рассмотрим этот способ на следующем примере. Пусть программа на языке С с использованием цикла *while* выглядит следующим образом:



```

1. int a = 1;
2. while(N)
3. {
4. N--;
5. a++;
6. }

```

На Прологе это будет выглядеть так:

```
prologwhile(0):-!.
```

```
prologwhile(N,A):-N1=N-1,A1=A+1, prologwhile(N1,A1).
```

**Второй способ.** Использование предиката, который можно передо-  
казать, и предиката fail

Функция стандартного предиката fail заключается в том, что он осу-  
ществляет вынужденное неудачное завершение выполнения предиката,  
в теле которого он находится, и, таким образом, инициирует бэктрекинг.  
Рассмотрим организацию цикла с использованием предиката fail на сле-  
дующем примере.

Пусть заданы названия некоторых стран. Необходимо их выдать в  
той последовательности, в какой они хранятся в базе данных.

Пример организации выдачи всех альтернативных решений – пре-  
дикат fail

```
Predicates country(symbol)
```

```
Print_of_counuries
```

```
clauses
```

```
country (англия).
```

```
country (россия).
```

```
country (германия).
```

```
country (дания).
```

```
Print_of_counuries:-country(X), write(X), nl, fail.
```

```
Goal
```

```
Print_of_counuries.
```

Здесь nl – стандартный предикат, осуществляющий перевод курсора  
на следующую строку.

**Третий способ.** Использование предиката, который можно передо-  
казать, и логического утверждения

Например, необходимо выдавать на экран пары «предок» – «пото-  
мок» до тех пор, пока не встретится потомок с именем «Миша».

*?- predok(A, B),  
write(A), write(' является предком '), writeln(B),  
B = 'Миша'; true.*

**Ответ:**

*Вика является предком Коля  
Вася является предком Коля  
Коля является предком Юля  
Юля является предком Миша*

Такая конструкция соответствует циклу с постусловием.

**Четвертый способ.** Организация цикла со счетчиком с использованием предиката *repeat* и динамического добавления фактов в базу знаний.

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

*repeat.  
repeat:- repeat.*

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

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

## ЗАДАНИЕ НА ПРАКТИЧЕСКУЮ РАБОТУ

1. а) Напишите на visual prolog программу в виде проекта, использующую внутреннюю базу данных и позволяющую спрашивать у пользователя, каким языком он владеет, записывая при этом ответы во внутреннюю базу данных. За основу можно взять следующую схему, заменив предикаты READ и WRITE эквивалентными предикатами VISUAL PROLOG:

ЯЗЫК (итальянский).

ЯЗЫК (немецкий).

ЯЗЫК (японский).

ЯЗЫК (французский).

ЯЗЫК (английский).

ДИАЛОГ:- WRITE («Введите ваше имя:»),

READ (Имя), ЯЗЫК (Яз),

WRITE («Знаете ли вы»),

WRITE (Яз),

WRITE («язык»),

READ (да),

ASSERT (ВЛАДЕЕТ (Имя, Яз)),

FAIL;

б) в базу данных включите факты: ЯЗЫК(...), ВЛАДЕЕТ (\_, \_);

в) измените программу, включив в нее предикаты чтения базы данных из файла и записи в файл по окончании сеанса работы.

2. Разработайте индивидуальную программу согласно предлагаемым индивидуальным заданиям таким образом, чтобы все основные факты вашей программы хранились во внутренней базе данных (считывались из файла, обрабатывались программой и снова записывались в файл). При этом введите диалог с пользователем для добавления или удаления фактов (за основу организации диалога возьмите реализацию задания 1).

### Индивидуальные задания

1. Разработайте базу данных, содержащую сведения об игрушках, предлагаемых в магазине: название игрушки (кукла, кубики, мяч и т. д.), ее стоимость и возрастные границы детей, для которых игрушка

предназначена. Получите следующие сведения (примерный перечень, придумайте еще столько же):

а) название игрушек, цена которых не превышает 400 руб. и которые подходят детям пяти лет;

б) название игрушек, которые подходят как детям четырех лет, так и детям 10 лет;

в) цены всех кубиков;

г) можно ли дедушке подобрать внуку на день рождения игрушку, любую, кроме мяча, подходящую ребенку трех лет, и дополнительно мяч так, чтобы суммарная стоимость покупки не превышала 500 руб.?

д) название наиболее дорогих игрушек (цена которых отличается от самой дорогой игрушки не более чем на 100 руб.).

2. Сведения об ученике состоят из его имени и фамилии, пола, названия класса, отметок, полученных учениками в последней четверти. Спроектировать систему, отвечающую на главный вопрос: возьмет ли школа главный приз? Введены следующие ограничения.

- Школа возьмет главный приз, если в ней будет проведено три олимпиады, конкурс красоты, гонки, не будет беспорядков и число лентяев не более трех.

- Беспорядки происходят, если есть класс, где нет отличников и в нем более четырех человек.

- Гонки проводятся, если в них участвует не менее двух человек.

- Среди участников гонок могут быть только троечники.

- В гонках участвуют и девочки, и мальчики.

- На конкурсе красоты должно быть не менее двух участников и трех судей.

- Судьей не может быть человек, занятый в нескольких олимпиадах.

- Ученик, участвующий в олимпиаде по физике, не может участвовать в конкурсе красоты.

- Только девочки участвуют в конкурсе красоты.

- Ученик участвует в олимпиаде, если он отличник по этому предмету.

- Олимпиада по предмету проводится, если на каждой параллели не менее четырех участников, участвующих в олимпиаде.

3. Разработайте базу данных библиотеки с самыми типичными отношениями между объектами (поиск книг по автору, поиск автора по названию книги, должники и т. д.).

4. Разработать справочно-информационную систему авиакомпании. Система должна содержать базу данных со следующей информацией: номер рейса, пункт отправления, пункт назначения, тип самолета, время отправления, время прибытия, дни выполнения рейсов, цена билета. Система должна подбирать рейсы (с учетом стыковок рейсов в течение одних суток) по минимальной стоимости билетов.

5. Разработайте базу данных – лекарства. Необходимо учесть симптомы болезни, возраст больного, противопоказания, стоимость, наличие и т. д. Система должна помогать фармацевту подбирать покупателям лекарства без рецептов (по ряду симптомов и противопоказаний) и в соответствии с их финансовыми возможностями.

6. Разработайте базу данных – продукты (магазины, ассортимент, стоимость, фирма-производитель и т. д.). Система должна помогать пользователю в выборе необходимого набора качественных продуктов в магазинах города при ограниченных денежных средствах.

7. Разработайте базу данных – одежда (по аналогии с заданием варианте 1).

8. Разработайте базу данных – кафедра (списочный состав, читаемые курсы, время и место проведения занятий, взаимозаменяемость и т. д.).

9. По аналогии с заданием варианта 2 придумайте ограничения и спроектируйте систему для ответа на вопрос: попадет ли студенческая группа на слет лучших групп?

10. Спроектируйте систему, помогающую человеку, заботящемуся о своем весе и здоровье, определять число калорий в определенном наборе продуктов, выбранном для приготовления пищи на день, и подбирать набор продуктов, удовлетворяющий качественному составу пищи (белки, жиры, углеводы, витамины, микроэлементы).

11. Разработайте справочно-информационную систему железнодорожной компании. Система должна содержать базу данных со следующей информацией: номер рейса, пункт отправления, пункт назначения, тип вагона, время отправления, время прибытия, дни выполнения рейсов, цена билета.

Система должна подбирать рейсы (с учетом пересадок в течение одних суток) по минимальному времени в пути.

12. Разработайте базу данных – городской транспорт. Система должна подбирать наиболее удобные маршруты с учетом стыковки различных видов транспорта и маршрутов.

## БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Авдеенко Т.В.* Введение в искусственный интеллект и логическое программирование: учебное пособие. – Новосибирск: Изд-во НГТУ, 2007. – 64 с.
2. *Авдеенко Т.В.* Логическое программирование. Методические указания к выполнению лабораторных работ для студентов III курса ФПМИ / Т.В. Авдеенко, Т.А. Шапошникова. – Новосибирск: Изд-во НГТУ, 2002. – 44 с.
3. *Анисимов В.В.* Интеллектуальные информационные системы / <https://www.sites.google.com/site/anisimovkhv/learning/iis/lecture>.
4. *Братко И.* Программирование на языке ПРОЛОГ для искусственного интеллекта: пер. с англ. – М.: Мир, 1990. – 560 с., ил.
5. *Гаврилов А.В.* Основы программирования на Турбо-Прологе: учебное пособие / А.В. Гаврилов, Ю.В. Новицкая. – Новосибирск: Изд-во НГТУ, 1993. – 90 с.
6. *Доорс Дж.* Пролог – язык программирования будущего/ А.Р. Рейблейн, С. Вадера: пер. с англ.; предисловие А.Н. Волкова. – М.: Финансы и статистика, 1990. – 144 с., ил.
7. *Ковальски Р.* Логика в решении проблем: пер. с англ. – М.: Наука, 1990. – 278 с.
8. *Рассел С.* Искусственный интеллект: современный подход. – 2-е изд.: пер с англ. / С. Рассел, П. Норвиг. – М.: Издательский дом «Вильямс», 2006. 1408 с.
9. *Стерлинг Л., Шапиро Э.* Искусство программирования на языке Пролог / Л. Стерлинг, Э. Шапиро. – М.: Мир, 1987. – 333 с.
10. *Шрайнер П.А.* Основы программирования на языке Пролог / интернет-университет информационных технологий – ИНТУИТ.РУ, Серия: Основы информационных технологий, 2005. – 176 с.
11. *Янсон А.* Турбо-Пролог в сжатом изложении: пер. с нем. – М.: Мир, 1991. – 94 с., ил.
12. Visual Prolog. – Prolog Development Centre, [www.visual-prolog.com](http://www.visual-prolog.com).

## ОГЛАВЛЕНИЕ

<b>1. Общие сведения о языке программирования Пролог .....</b>	<b>3</b>
1.1. Введение в Пролог .....	6
1.2. Унификация .....	11
1.3. Рекурсия .....	12
1.4. Описание работы Пролога по поиску решения задачи .....	14
1.5. Механизм возврата в Прологе .....	18
<b>2. Разработка простейшей базы знаний на языке Пролог .....</b>	<b>20</b>
2.1. Синтаксис и семантика (смысл) логической программы .....	20
2.2. Анализ предметной области .....	29
2.2.1. Построение генеалогического дерева .....	29
2.2.2. Перевод генеалогического дерева в базу фактов .....	30
2.2.3. Составление правил искомых родственных связей в базе знаний ....	32
2.2.4. Примеры запросов к базе знаний и ее ответов .....	32
<b>3. Разработка программ на платформе VISUAL PROLOG .....</b>	<b>34</b>
3.1. Основные начальные сведения о PROLOG и VISUAL PROLOG .....	34
3.2. Объявления и определения в VISUAL PROLOG .....	37
3.2.1. Ключевые слова .....	37
3.2.2. Понятие цели в VISUAL PROLOG .....	39
3.2.3. Файловая структура программы, написанной в среде VISUAL PROLOG .....	40
3.2.4. Проблемы доступа к программам большого объема .....	40
3.3. Стандартные предикаты ввода-вывода в VISUAL PROLOG .....	42
3.4. Пример завершенной программы на VISUAL PROLOG .....	42
3.5. Создание консольного проекта в среде VISUAL PROLOG IDE .....	44
<b>4. Внутренняя база данных VISUAL PROLOG .....</b>	<b>49</b>
4.1. Способы создания и обработки внутренней базы данных в VISUAL PROLOG .....	49
4.2. Способы организации циклов в VISUAL PROLOG .....	56
Задания на практическую работу .....	59
Библиографический список .....	62

**Авдесенко Татьяна Владимировна  
Целебровская Марина Юрьевна**

**ВВЕДЕНИЕ В ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ  
И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ  
ПРОГРАММИРОВАНИЕ В СРЕДЕ VISUAL PROLOG**

**Учебное пособие**

Редактор *И.Л. Кескевич*  
Выпускающий редактор *И.П. Брованова*  
Корректор *И.Е. Семенова*  
Дизайн обложки *А.В. Ладыжская*  
Компьютерная верстка *Л.А. Веселовская*

Налоговая льгота – Общероссийский классификатор продукции  
Издание соответствует коду 95 3000 ОК 005-93 (ОКП)

---

Подписано в печать 08.06.2020. Формат 60 × 84 1/16. Бумага офсетная. Тираж 80 экз.  
Уч.-изд. л. 3,72. Печ. л. 4,0. Изд. № 8. Заказ № 636. Цена договорная

---

Отпечатано в типографии  
Новосибирского государственного технического университета  
630073, г. Новосибирск, пр. К. Маркса, 20