Введение в Пролог

В октябре 1981 года Японское министерство международной торговли и промышленности объявило о создании исследовательской организации — Института по разработке методов создания компьютеров нового поколения (Institute for New Generation Computer Technology Research Center).

Целью данного проекта было создание систем обработки информации, базирующихся на знаниях.

Предполагалось, что эти системы будут обеспечивать простоту управления за счет возможности общения с пользователями при помощи естественного языка.

Эти системы должны были самообучаться, использовать накапливаемые в памяти знания для решения различного рода задач, предоставлять пользователям экспертные консультации, причем от пользователя не требовалось быть специалистом в информатике.

Предполагалось, что человек сможет использовать ЭВМ пятого поколения так же легко, как любые бытовые электроприборы типа телевизора, магнитофона и пылесоса. Вскоре вслед за японским стартовали американский и европейский проекты.

- Появление таких систем могло бы изменить технологии за счет использования баз знаний и экспертных систем. Основная суть качественного перехода к пятому поколению ЭВМ заключалась в переходе от обработки данных к обработке знаний. Японцы надеялись, что им удастся не подстраивать мышление человека под принципы функционирования компьютеров, а приблизить работу компьютера к тому, как мыслит человек, отойдя при этом от фон неймановской архитектуры компьютеров. В 1991 году предполагалось создать первый прототип компьютеров пятого поколения

В качестве основной методологии разработки программных средств для проекта ЭВМ пятого поколения было избрано логическое программирование, ярким представителем которого является *язык Пролог*.

В настоящее время Пролог остается наиболее популярным языком искусственного интеллекта в Японии и Европе (в США, традиционно, более распространен другой язык искусственного интеллекта —язык функционального программирования Лисп).

- Название *языка "Пролог"*происходит от слов ЛОГическое ПРОграммирование (PROgrammation en LOGique во французском варианте и PROgramming in LOGic — в английском).
- Пролог основывается на таком разделе математической логики, как исчисление предикатов.

- Традиционно под программой понимают последовательность операторов (команд, выполняемых компьютером). Этот стиль программирования принято называть **императивным**. Программируя в императивном стиле, программист должен объяснить компьютеру, **как** нужно решать задачу.
- Противоположный ему стиль программирования — так называемый **декларативный стиль**, в котором программа представляет собой совокупность утверждений, описывающих фрагмент предметной области или сложившуюся ситуацию. Программируя в декларативном стиле, программист должен описать, **что** нужно решать.

- Соответственно и *языки* программирования делят на *императивные* и *декларативные*.
- ***Императивные языки*** основаны на фон неймановской модели вычислений компьютера. Решая задачу, императивный программист вначале создает модель в некоторой формальной системе, а затем переписывает решение на *императивный язык* программирования в терминах компьютера. Но, во-первых, для человека рассуждать в терминах компьютера довольно неестественно. Во-вторых, последний этап этой деятельности (переписывание решения на язык программирования) по сути дела не имеет отношения к решению исходной задачи. Очень часто императивные программисты даже разделяют работу в соответствии с двумя описанными выше этапами. Одни люди, постановщики задач, придумывают решение задачи, а другие, кодировщики, переводят это решение на язык программирования.

- В основе ***декларативных языков*** лежит формализованная человеческая логика. Человек лишь описывает решаемую задачу, а поиском решения занимается императивная система программирования. В итоге получаем значительно большую скорость разработки приложений, значительно меньший размер исходного кода, легкость записи знаний на *декларативных языках*, более понятные, по сравнению с *императивными языками*, программы.

Основные области применения пролога:

- быстрая разработка прототипов прикладных программ;
- автоматический перевод с одного языка на другой;
- создание естественно-языковых интерфейсов для существующих систем;
- символьные вычисления для решения уравнений, дифференцирования и интегрирования;
- проектирование динамических реляционных баз данных;
- экспертные системы и оболочки экспертных систем;
- автоматизированное управление производственными процессами;
- автоматическое доказательство теорем;
- полуавтоматическое составление расписаний;
- системы автоматизированного проектирования;
- базирующееся на знаниях программное обеспечение;
- организация сервера данных или, точнее, сервера знаний, к которому может обращаться клиентское приложение, написанное на каком-либо языке программирования

Области, для которых Пролог не предназначен:

- большой объем арифметических вычислений (обработка аудио, видео и т. д.);
- написание драйверов.

Имена конкретных объектов, отношений, свойств образуются по определенным правилам, зависящим от версии языка Пролог, и называются **символическими именами.**

- ***Символическое имя (атом)*** *– это неразрывная цепочка букв, цифр и символа подчеркивания, начинающаяся со строчной латинской (русской) буквы.*

Для **именования объектов и их свойств** могут использоваться строки – последовательности любых символов, заключенные в двойные кавычки.

- Например: стол, студент,table_1, «группа 408».

Для описания в программе некоторого объекта, принадлежащего определенному классу, используется **переменная.**

- ***Переменная*** *в программе представляется своим именем. Имя переменной в Прологе – это латинских букв или цифр, начинающихся с прописной латинской буквы или символа подчеркивания. Примеры имен переменных: D, X,Y,_P, Leda*

Чтобы описать отношение, необходимо указать его имя и перечислить либо классы объектов, либо конкретные объекты, связываемые этим отношением

<имя отношения> (<имя объекта1>, <имя объекта2>, …,< имя объекта n>)

Для описания отношений в программе на Прологе используются предикаты.

Предикат – это логическая функция от **n** аргументов, имеющая только два значения «истина» или «ложь»:

<имя предиката (<аргумент1>, <аргумент2>, …,< аргумент n>)

Знания о предметной области выражаются на языке Пролог в виде предложений, называемых **утверждениями** (CLAUSES).

<заголовок>. /*факт*/

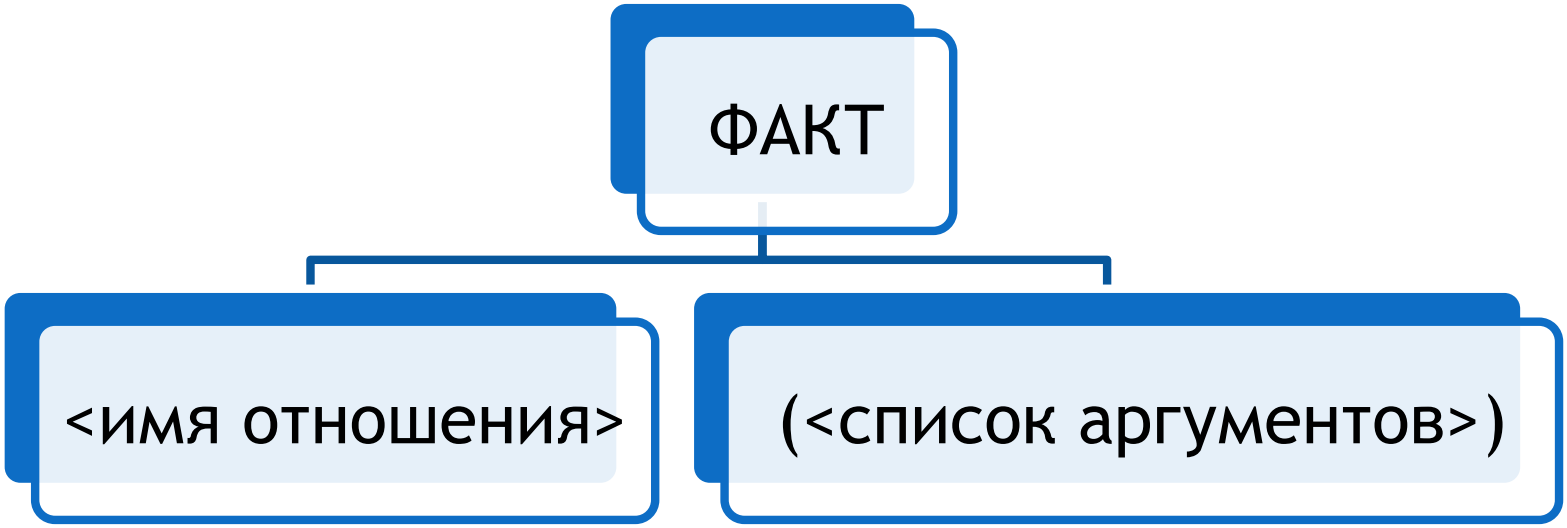
или

<заголовок>:-<тело>. /*правило*/

где ***заголовок*** является предикатом и полностью характеризует описываемое отношение.

Тело утверждения состоит либо из одного предиката либо из списка предикатов, разделенных знаками “,” “;”, “not”, соответствующими логическим операциям И, ИЛИ, НЕ .

```
DOMAINS /*описание типов данных*/
  name = string
FACTS    /*описание динамической базы данных*/
закуска (name)
мясо (name)
рыба (name)
десерт (name)
CLAUSES /*утверждения (факты и правила) БЗ*/
закуска («артишоки_в_белом_соусе»).
закуска («трюфели_в_шампанском»).
закуска («салат_с_яйцом»).
мясо («говяжье_жаркое»).
мясо («цыпленок_в_соусе»).
рыба («окунь_во_фритюре»).
рыба(«фаршированный_судак»).
десерт(«грушевое_мороженое»).
```


ФАКТ
<имя отношения>
(<список аргументов>)

Программа начинает выполняться, если в неё ввести те вопросы, ответы на которые хочет получить пользователь. Для этого предназначен раздел **GOAL** (Цель).

В нем записываются необходимые вопросы – третий тип утверждений в программе на Прологе

GOAL

рыба (*окунь_во_фритюре*).

- yes

GOAL

закуска (X).

X= артишоки_в_белом_соусе

X= трюфели_в_шампанском

X= салат_с_тунцом

```
PREDICATES /*Определение предиката блюдо*/
блюдо (name)
CLAUSES
блюдо(Y):-мясо(Y)
блюдо(Y):-рыба (Y)
```

GOAL

блюдо(Y).

Y= **говяжье_жаркое**

Y= **цыпленок_в_соусе**

Y= **окунь_во_фритюре**

Y= **фаршированный_судак**

/*Определение отношения «обед»*/

обед (X,Y,Z):- закуска (X), блюдо(Y), десерт(Z).

GOAL

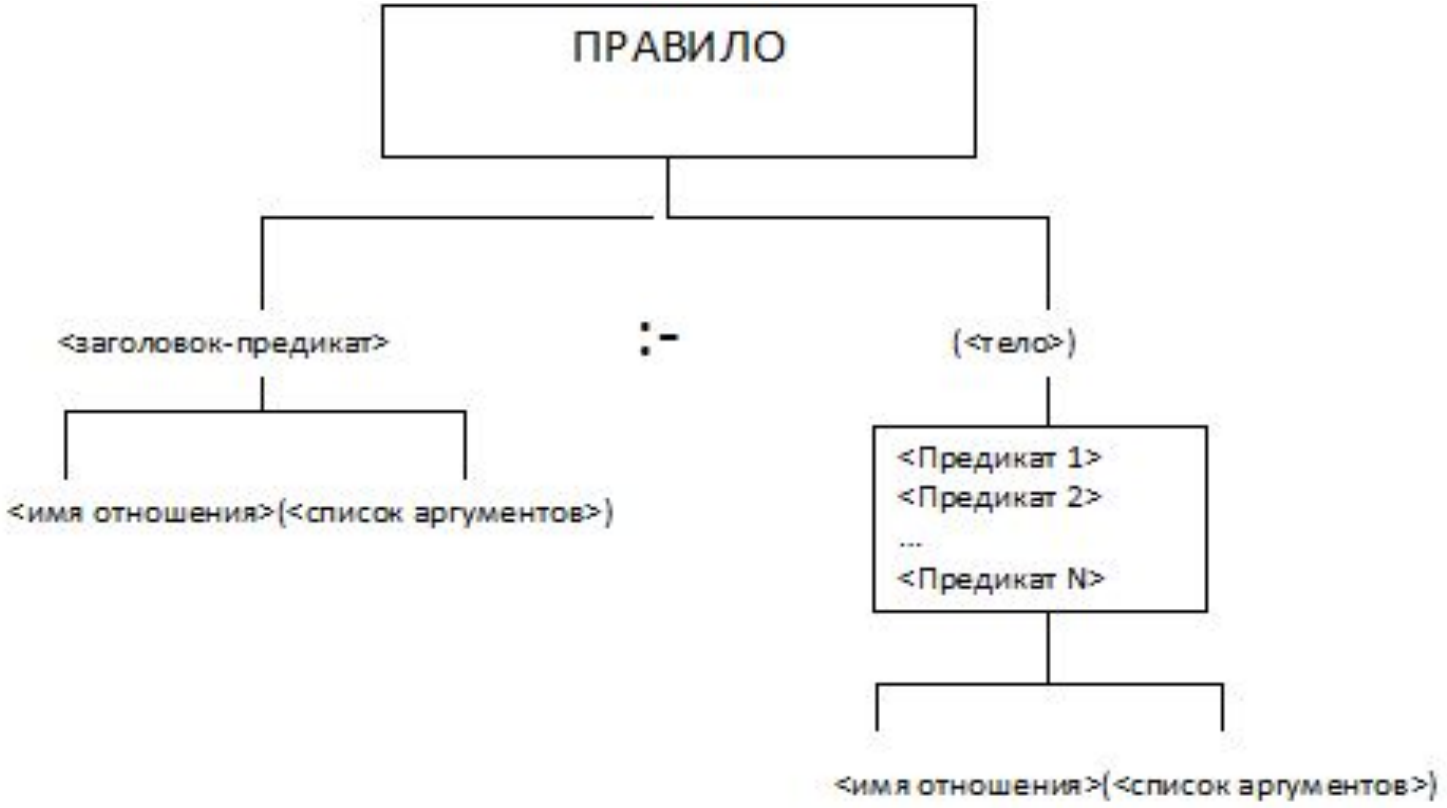
обед (X,Y, Z).

программа выдаст список всех возможных комбинаций из трех блюд:

X = артишоки_в_белом_соусе, Y = говяжье_жаркое, Z= грушевое_мороженое …

GOAL

обед (X,Y, Z), рыба (Y).


ПРАВИЛО
<заголовок-предикат>
:-
(<тело>)
<имя отношения>(<список аргументов>)
<Предикат 1>
<Предикат 2>
...
<Предикат N>
<имя отношения>(<список аргументов>)

ПРИМЕРЫ ПРАВИЛ:

любит (X, «баскетбол»):- любит (X, «бег»).

можно купить (X) :-есть в магазине(X,V),V<3000.

старше (P_1,P_2) :-возраст (P_1, V_1), возраст (P_2, V_2), $V_1 > V_2$.

Варианты студентов

- Нравится(X, "пироженое»):-любит (X, «сладкое)
- Выше (X1,X2):-рост(X1,У1), рост (X2,У2), У1>У2
- Нужно закупать(x):-есть на складе(x),x<10
- Уложился в норматив(x,y):-время(x,y),y<12
- Может(x,”скачатьфильм”):-Есть(x,” интернет”)
- ПерекинутьФайл(x1,x2):-ЕмкостьНосителя (x1,y1),РазмерФайла(x2,y2),У1>У2

ВАРИАНТЫ СТУДЕНТОВ

- Нравится(X, “Mercedes”):-нравится(X,” немецкий автопром”)

- Опытнее(X1,X2):-стаж(X1,V1), стаж(X2, V2), V1>V2

```
DOMAINS /*описание типов данных*/
  kol_vo = integer
FACTS    /*описание динамической базы данных*/
калории (name, kol_vo )
CLAUSES /*утверждения (факты и правила) БЗ*/
калории(«артишоки_в_белом_соусе»,     150).
калории («трюфели_в_шампанском»,      212 ).
калории («салат_с_яйцом»,          202).
калории («говяжье_жаркое»,      532).
калории («цыпленок_в_соусе»,  400).
калории («окунь_во_фритюре»,   270).
калории(«фаршированный_судак»,   254).
калории(«грушевое_мороженое»,   223).
калории («земляника _со_сливками»,  289).
```

- **GOAL**

закуска (X), калории(X,Y).

```
PREDICATES
значение (name, name, name, kol_vo)
сбалансированный_обед(name, name, name)
CLAUSES /*Определение отношения «калорийность обеда»*/
значение (X,Y,Z,V):- калории(X,E),
                     калории(Y, P),
                     калории(Z,D),
                      V=E+P+D
сбалансированный_обед (X,Y,Z):-обед (X,Y, Z), значение (X,Y,Z,V),V<800.

GOAL

сбалансированный_обед (X,Y,Z).
```

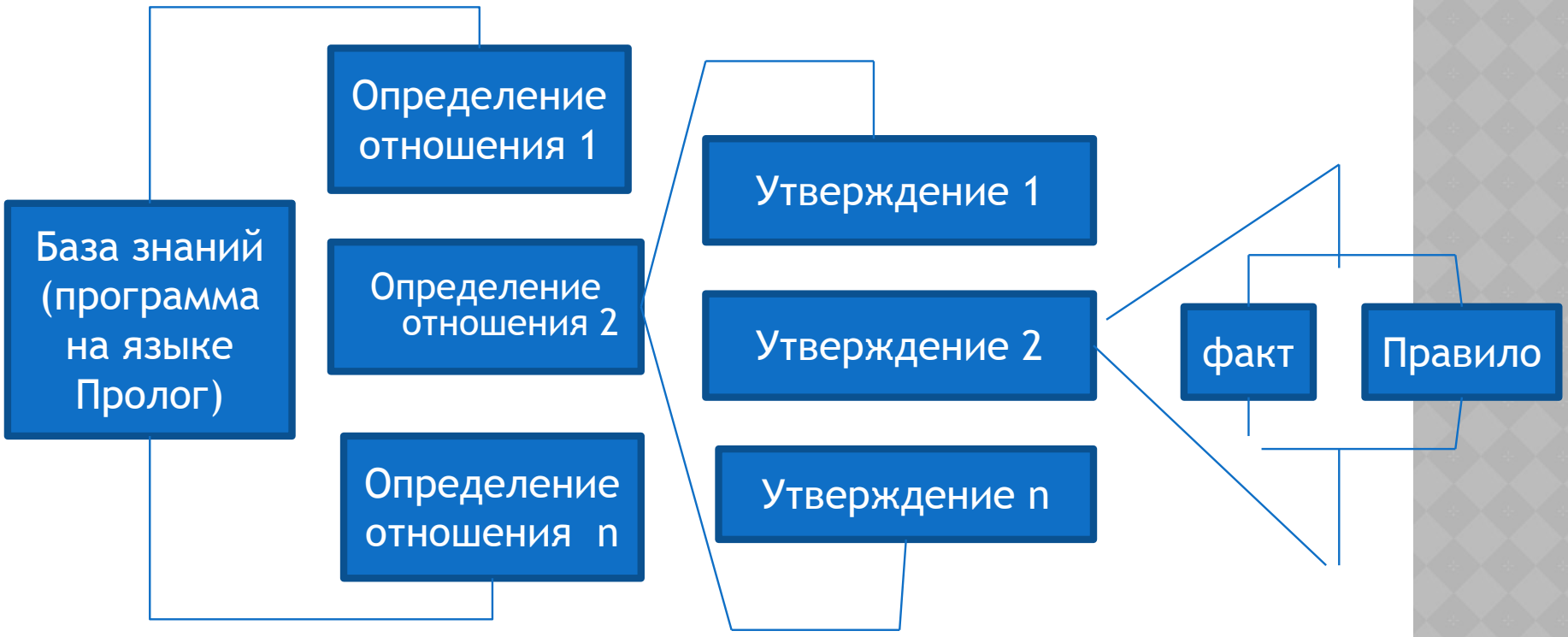
DOMAINS /*описание типов данных*/
FACTS (DATABASE) /*описание динамической базы данных*/

PREDICATES /*Определение предикатов*/

CLAUSES /*утверждения (факты и правила) БЗ*/

GOAL

СТРУКТУРА РАЗДЕЛА УТВЕРЖДЕНИЙ ПРОГРАММЫ



symbol – символическое имя

string - строка

char - отдельный символ, заключенный в апострофы

Integer целое число в диапазоне от -32768 до 32767

byte - целое число в диапазоне от 0 до 255

word целое число в диапазоне от 0 до 65535

real- любое число

Пять основных операций определяют реляционную алгебру:

1. Объединение
2. Симметрическая разность
3. Декартово произведение
4. Проекция
5. Выборка

ОБЪЕДИНЕНИЕ

r_unon_s(X1,X2,…, Xn) :- r(X1,X2,…, Xn).
r_unon_s(X1,X2,…, Xn) :- s(X1,X2,…, Xn).

Пример:
блюдо(Y):-мясо(Y).
блюдо(Y):-рыба (Y).

Симметрическая разность

r_diff_s(X1,X2,…, Xn) :- r(X1,X2,…, Xn), not s(X1,X2,…, Xn).

r_diff_s(X1,X2,…, Xn) :- s(X1,X2,…, Xn), not r(X1,X2,…, Xn).

Пример:

футболист_либо_волейболист(X) :- футболист(X), not волейболист(X).

футболист_либо_волейболист() :- волейболист(X), not футболист(X)

ДЕКАРТОВО ПРОИЗВЕДЕНИЕ

r_x_s(X1,X2,…,Xn, Y1,Y2,…,Yn):- r(X1,X2,…Xn), s(Y1,Y2,…,Yn).

Пример:

/*Определение отношения «обед»*/

обед (X,Y,Z):- закуска (X), блюдо(Y), десерт (Z).

ПРОЕКЦИЯ

r7(X1, X3):- r(X1,X2, …, Xn).

Пример:

аптека(6, «Цвиллинга, 6», «264-67-66»).

телефон_аптеки(NomApt,TelApt):- аптека (NomApt, _, TelApt).

ВЫБОРКА

rv(X1, X3):- r(X1,X2,X3), X2>X1

Пример:

аптека(6, «Цвиллинга, 6», «264-67-66»).

телефон_аптеки(NomApt,TelApt):- аптека (NomApt, _, TelApt), NomApt <8.

сбалансированный_обед (X,Y,Z):-обед (X,Y, Z), значение (X,Y,Z,V),V<800.

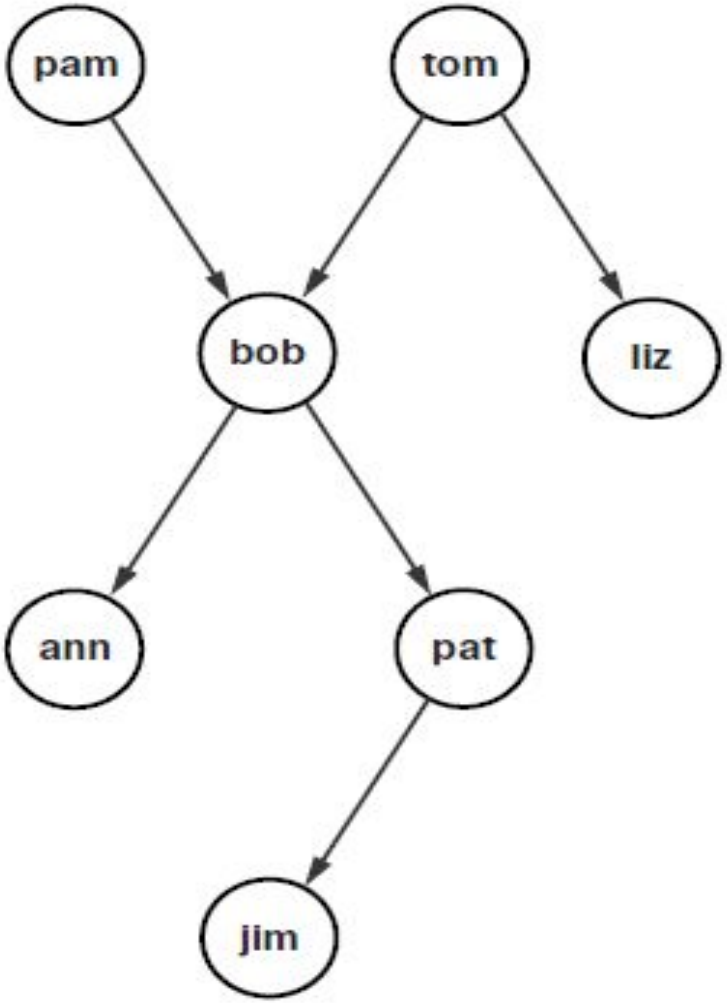


Рис. 1.1. Дерево семейных отношений

- name, child, grandchild – string

- Parent(name, child)
- nondetermed Grandparent(name, grandchild)

- Parent(Pam,Bob)
- Parent(Tom,Bob)
- Parent(Tom,Liz)
- Parent(Bob,Ann)
- Parent(Bob,Pat)
- Parent(Pat,Jim)

- Grandparent(Pam, Ann)
- Grandparent(Tom, Ann)
- Grandparent(Bob, Jim)
- Grandparent(Pam, Pat)
- Grandparent(Tom,Pat
-)

Prolog — это язык программирования для символических, нечисловых вычислений. Он особенно хорошо приспособлен для решения проблем, которые касаются объектов и отношений между объектами. На рис. 1.1 приведен подобный пример: семейные отношения. Тот факт, что Том является одним из родителей Боба, можно записать на языке Prolog следующим образом:

```
parent( tom, bob).
```

В данном случае в качестве имени отношения выбрано слово `parent`; `tom` и `bob` являются параметрами этого отношения. По причинам, которые станут понятными позже, такие имена, как `tom`, записываются со строчной буквой в начале. Все дерево семейных отношений, показанное на рис. 1.1, определено с помощью следующей программы Prolog:

```
parent( pam, bob).
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
parent( pat, jim).
```

Эта программа состоит из шести предложений, каждое из которых объявляет один факт об отношении `parent`. Например, факт `parent( tom, bob)` представляет собой конкретный экземпляр отношения `parent`. Такой экземпляр называют также *связью*. В целом *отношение* определяется как множество всех своих экземпляров.

После передачи соответствующей программы в систему Prolog последней можно задать некоторые вопросы об отношении parent, например, является ли Боб одним из родителей Пэт? Этот вопрос можно передать системе Prolog, введя его на терминале:

```
?- parent( bob, pat).
```

Обнаружив, что это — факт, о существовании которого утверждается в программе, Prolog отвечает:

```
yes
```

После этого можно задать еще один вопрос:

```
?- parent( liz, pat).
```

Система Prolog ответит:

```
no
```

поскольку в программе нет упоминания о том, что Лиз является одним из родителей Пэт. Система ответит также “no” на вопрос

```
?- parent( tom, ben).
```

поскольку в программе даже не встречалось имя Бэн.

Кроме того, системе можно задать более интересные вопросы. Например, кто является родителями Лиз?

```
?- parent( X, liz).
```

На этот раз Prolog ответит не просто “yes” или “no”, а сообщит такое значение X, при котором приведенное выше утверждение является истинным. Поэтому ответ будет таковым:

```
X = tom
```

Вопрос о том, кто является детьми Боба, можно сообщить системе Prolog следующим образом:

```
?- parent( bob, X).
```

На этот раз имеется больше одного возможного ответа. Система Prolog вначале выдаст в ответ одно решение:

```
X = ann
```

Теперь можно потребовать у системы сообщить еще одно решение (введя точку с запятой), и Prolog найдет следующий ответ:

```
X = pat
```

Если после этого будут затребованы дополнительные решения, Prolog ответит “no”, поскольку все решения уже исчерпаны.

Этой программе может быть задан еще более общий вопрос о том, кто является чьим родителем? Этот вопрос можно также сформулировать иным образом:

```
Найти X и Y, такие, что X является одним из родителей Y.
```

Этот вопрос может быть оформлен на языке Prolog следующим образом:

```
?- parent( X, Y).
```

После этого Prolog начнет отыскивать все пары родителей и детей одну за другой. Решения отображаются на дисплее по одному до тех пор, пока Prolog получает указание найти следующее решение (в виде точки запятой) или пока не будут найдены все решения. Ответы выводятся следующим образом:

```
X = pam
Y = bob;
X = tom
Y = bob;
X = tom
Y = liz;
```

Чтобы прекратить вывод решений, достаточно нажать клавишу <Enter> вместо точки с запятой.

Этой программе, рассматриваемой в качестве примера, можно задать еще более сложный вопрос, например, спросить о том, кто является родителями родителей Джима (дедушками и бабушками). Поскольку в программе непосредственно не предусмотрено использование соответствующего отношения `grandparent`, этот запрос необходимо разбить на следующие два этапа (рис. 1.2).

1. Кто является одним из родителей Джима? Предположим, что это — некоторый объект `Y`.
2. Кто является одним из родителей `Y`? Предположим, что это — некоторый объект `X`.

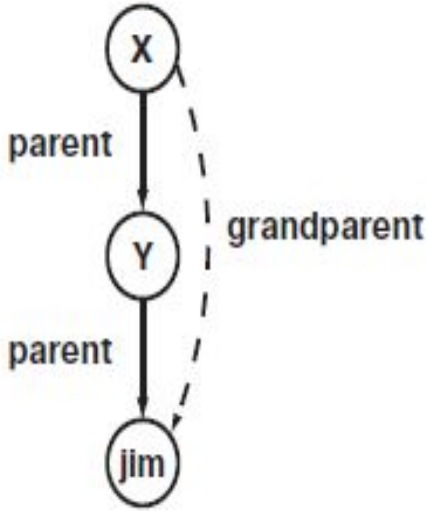


Рис. 1.2. Отношение `grandparent`, выраженное как композиция двух отношений `parent`

Подобный сложный запрос записывается на языке Prolog как последовательность двух простых:

```
?- parent( Y, jim), parent( X, Y).
```

Ответ должен быть следующим:

```
X = bob
Y = pat
```

Этот составной запрос можно прочитать таким образом: найти такие X и Y, которые удовлетворяют следующим двум требованиям:

```
parent( Y, jim) и parent( X, Y)
```

Если же будет изменен порядок следования этих двух требований, то логический смысл всего выражения останется тем же:

```
parent( X, Y) и parent( Y, jim)
```

Безусловно, такое же действие может быть выполнено и в программе Prolog, поэтому запрос:

```
?- parent( X, Y), parent( Y, jim).
```

выдаст тот же результат.

Аналогичным образом, программе можно задать вопрос о том, кто является внуками Тома:

```
?- parent( tom, X), parent( X, Y).
```

Система Prolog ответит следующим образом:

```
X = bob
Y = ann;
X = bob
Y = pat
```

Могут быть также заданы и другие вопросы, например, имеют ли Энн и Пэт общих родителей. Эту задачу также можно разбить на два этапа.

1. Кто является одним из родителей Энн (X)?
2. Является ли (тот же) X одним из родителей Пэт?

Поэтому соответствующий вопрос в языке Prolog выглядит следующим образом:

```
?- parent( X, ann), parent( X, pat).
```

Ответом на него является:

```
X = bob
```

Рассматриваемый пример программы позволяет проиллюстрировать перечисленные ниже важные понятия.

- В языке Prolog можно легко определить отношение, такое как `parent`, задавая *n*-элементные кортежи объектов, которые удовлетворяют этому отношению.
- Пользователь может легко запрашивать систему Prolog об отношениях, определенных в программе.
- Программа Prolog состоит из предложений. Каждое предложение оканчивается точкой.
- Параметрами отношений могут быть (кроме всего прочего) определенные объекты, или константы (такие как `tom` и `ann`), а также объекты более общего характера (такие как `X` и `Y`). Объекты первого типа, применяемые в рассматриваемой программе, называются *атомами*. Объекты второго типа называются *переменными*.

- Вопросы к системе состоят из одной или нескольких *целей*. Последовательность целей, такая как

```
parent( X, ann), parent( X, pat)
```

означает конъюнкцию целей:

```
X является одним из родителей Энн и
X является одним из родителей Пэт.
```

Слово "цель" (goal) используется для обозначения таких вопросов потому, что система Prolog воспринимает вопросы как цели, которых необходимо достичь.

- Ответ на вопрос может быть положительным или отрицательным, в зависимости от того, может ли быть достигнута соответствующая цель или нет. В случае положительного ответа считается, что соответствующая цель была достижимой и что цель достигнута. В противном случае цель была недостижимой и не достигнута.
- Если вопросу соответствует несколько ответов, Prolog отыскивает столько ответов, сколько потребует пользователь (в пределах возможного).

1.1. При условии, что определено отношение `parent`, как описано в этом разделе (см. рис. 1.1), укажите, какой ответ даст система Prolog на приведенные ниже вопросы?

а) `?- parent( jim, X).`

б) `?- parent( X, jim).`

в) `?- parent( pam, X), parent( X, pat).`

г) `?- parent( pam, X), parent( X, Y), parent( Y, jim).`

1.2. Сформулируйте на языке Prolog перечисленные ниже вопросы об отношении `parent`.

а) Кто является родителем Пэт?

б) Имеет ли Лиз ребенка?

в) Кто является дедушкой или бабушкой Пэт?

Рассмотрим следующие предложения на естественном языке:

- «Маше нравятся цветы»
- «Аня любит сына»
- Сын Ани любит Машу
- «Оля любит всех, кого любит Аня»
- «Нам нравится всё, что нравится человеку, которого мы любим»

Как можно записать эти предложения в виде фактов ?

- нравится (маша, цветы)
- любит (аня, сын(аня))
- любит (сын(аня), маша)

- Четвертое предложение задает правило:

«Для любого X, если *любит (аня, X)*, то из этого следует, что *любит (оля, X)*»

Запишем это правило в виде импликации:

любит (оля, X) ▫ *любит (аня, X)*,

- Антон, Миша и Женя – члены альпинклуба. Каждый член альпинклуба или горнолыжник, или скалолаз или и то, и другое. Никто из скалолазов не любит дождь. Все горнолыжники любят снег. Миша любит всё, что не любит Антон, и не любит всё, что любит Антон. Антон любит снег и дождь. Есть ли член альпинклуба, который является скалолазом и не является горнолыжником? И Кто он?

ВАРИАНТ ИВАНА ПИМЕНОВА

- DOMAINS
- Name string
- PREDICATES
- Горнолыжник (Name)
- Скалолаз (Name)
- Любит_дождь (Name)
- Любит_снег (Name)
- FACTS
- горнолыжник (Name):-любит_снег (Name)
- скалолаз (Name):- not любит_дождь (Name)
- CLAUSES
- Любит_снег(Антон)
- Любит_дождь(Антон)
- Любит_снег(Миша):- not Любит_снег(Антон)
- Любит_дождь(Миша):- not Любит_дождь(Антон)
- Not Любит_снег(Миша):- Любит_снег(Антон)
- Not Любит_дождь(Миша):- Любит_дождь(Антон)
- GOAL
- ? Скалолаз(Name), not горнолыжник(name)

```
DOMAINS /*задание областей данных*/
  Name = anton;misha;jenia /*Область Name
   это или anton или misha или jenia */
   Weather = rain;snow /*Определение области
   Weather */
   Spec = skalolaz;gornolysnik /*Определение
   области Spec */
```

PREDICATES /*Определение предикатов*/

club (Name) /* Name является членом клуба*/

nlike (Name, Weater) /* Name не нравится Weater */

like (Name, Weater) /* Name нравится Weater */

is (Name, Spec) /* Name по специализации Spec */

question /*Теорема, которую нужно доказать*/

ФАКТЫ И ПРАВИЛА

- CLAUSES
- club(anton). club(misha). club(jenia).
- like(anton, snow). like(anton, rain).

- like(misha,X) :- not(like(anton,X)).

- nlike(X,Y):-not(like(X,Y)).
- nlike(misha,Y):-like(anton,Y).

- is (X, gornolysnik):- club(X), like(X,snow).
- is (X, skalolaz):-club(X), not(like(X,rain)).

Выясняем имя того, кто является скалолазом и не является горнолыжником. Предикат неудачи – fail , заставляет перейти к следующему определению question

```
question:-
   readchar(),
   nl, is(Name, skalolas),not(is(Name,gornolysnik)),
   cursor(10,20), write(“Скалолаз,но не горнолыжник),
   wrute (Name), readchar(),nl,fail.
```

Выясняем имя того, кто является и скалолазом и горнолыжником.

```
question:-
   nl, is(Name, skalolas),is(Name,gornolysnik),
   cursor(12,20), write("Скалолаз и горнолыжник: "),
   wrute (Name), readchar(),nl,fail.
```

- Какая специализация у Антона

```
question:-
   nl, is(anton, Spec),
   cursor(14,20), write("anton -", Spec),
   readchar(),nl,fail.
```

```
question:-
   nl, is(Name, gornolysnik),
   not (is(Name,skalolas)),
   cursor(16,20), write(“Не скалолаз, но
   горнолыжник: “),
   wrute (Name), readchar(),nl,fail.

GOAL
 question.
```

КОРОЛЬ ДУМАЕТ, ЧТО КОРОЛЕВА ДУМАЕТ, ЧТО ОНА НЕ В СВОЕМ УМЕ. В СВОЕМ ЛИ УМЕ КОРОЛЬ?

```
DOMAINS /*Описание областей данных*/
  Name = KL; KWA
    Swoystwo = wume;newume
 /* Swoystwo может принимать значение wume или
   newume* /
  Mysly = imeet_mesto(Swoystwo, Swoystwo)
  /* Mysly - это то, о чем можно думать*/
```

```
PREDICATES  /*Описание предикатов*/
    mojetbyt (Swoystwo)    /* mojetbyt  - свойство,
    принимающее значение из области Swoystwo */
    imeet_mesto(Swoystwo, Swoystwo)
    dumaet (Swoystwo, Mysly)
    wopros
```

```
CLAUSES  /*Описание  предметной области*/

 mojetbyt (wume). mojetbyt (newume).

/*можно быть или « в своем уме»
или «не в своем уме»*/
```

```
dumaet (wume, imeet_mesto(X,Y)):-
                    imeet_mesto(X,Y).
/*»в своем уме» думает то, что есть на самом
  деле*/
```

```
dumaet(newume, imeet_mesto(X,Y)):-not
  imeet_mesto(X,Y)).
/*» не в своем уме» думает то, чего нет на
  самом деле*/
```

```
imeet_mesto(X,Y). /*на самом деле, есть то,
  что есть*/
```

```
wopros:- mojetbyt(KL), mojetbyt(KWA),
 dumaet (KL, dumaet(KWA,
  imeet_mesto(KWA, newume))),
 nl, write («Королева -», KWA, «король - »,
  KL), nl, fail.
```

GOAL /*Запрос или теорема, которую нужно доказать*/

- wopros