Часть I
Язык Prolog

В этой части...

Глава 1

Введение в Prolog

В этой главе...

В этой главе описаны основные механизмы языка Prolog на примере многочисленных программ. Изложение материала является в основном неформальным, но в нем представлены многие важные концепции, такие как предложения, факты, правила и процедуры Prolog. Кроме того, рассматривается встроенный механизм перебора с возвратами системы Prolog и описываются различия между декларативным и процедурным значениями программы.

1.1. Определение отношений на основе фактов

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.

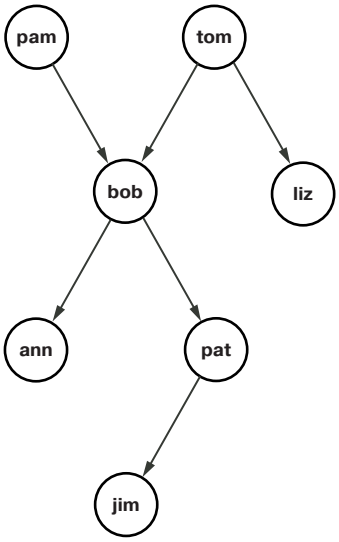

Рис. 1.1. Дерево семейных отношений

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`.

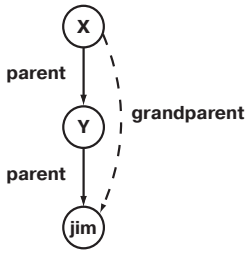

Рис. 1.2. Отношение `grandparent`, выраженное как композиция двух отношений `parent`

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 слу-

чае 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).

Упражнения

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?

1.2. Определение отношений на основе правил

Рассматриваемый пример программ можно легко дополнить с применением многих интересных способов. Вначале введем информацию о мужском или женском поле людей, участвующих в отношении `parent`. Эту задачу можно решить, добавив следующие факты к программе:

```
female( pam).
male( tom).
male( bob).
female( liz).
female( pat).
female( ann).
male( jim).
```

В этом случае введены отношения `male` и `female`. Эти отношения являются унарными (или одноместными). Бинарные отношения типа `parent` определяют связь между парами объектов. С другой стороны, унарные отношения могут использоваться для объявления простых свойств объектов, которые они могут иметь или не иметь. Первое из приведенных выше унарных предложений можно прочитать таким образом: Пэм — женщина. Вместо этого информацию, объявленную в двух унарных отношениях, можно передать с помощью одного бинарного отношения. В таком случае приведенная выше часть программы примет примерно такой вид:

```
sex( pam, feminine).
sex( tom, masculine).
sex( bob, masculine).
```

В качестве следующего дополнения к программе введем отношение `offspring` (отпрыск), обратное отношению `parent`. Отношение `offspring` можно определить таким же образом, как и `parent`, предоставив список обычных фактов об отношении `offspring`, и в качестве каждого факта указать такую пару людей, что один из них является сыном или дочерью другого, например:

```
offspring( liz, tom).
```

Но отношение offspring можно определить гораздо более изящно, используя то, что оно является противоположным parent и что parent уже было определено. Этот альтернативный способ может быть основан на следующем логическом утверждении:

```
Для всех X и Y,
  Y является сыном или дочерью X, если
  X является родителем Y.
```

Такая формулировка уже более близка к синтаксису языка Prolog. Соответствующее предложение Prolog, которое имеет тот же смысл, выглядит таким образом:

```
offspring( Y, X) :- parent( X, Y).
```

Это предложение можно также прочитать так:

```
Для всех X и Y,
  если X является родителем Y, то
  Y является сыном или дочерью X.
```

Предложения Prolog, такие как

```
offspring( Y, X) :- parent( X, Y).
```

называются *правилами*. Между фактами и правилами существует важное различие. Такие факты, как

```
parent( tom, liz).
```

представляют собой логические утверждения, которые всегда и безусловно являются истинными. С другой стороны, правила представляют собой утверждения, которые становятся истинными, если удовлетворяются некоторые условия. Поэтому принято считать, что правила состоят из следующих частей:

- условие (правая часть правила);
- заключение (левая часть правила).

Часть, соответствующая заключению, называется также *головой предложения*, а часть, соответствующая условию, — *телом предложения*, как показано ниже.

```
offspring( Y, X)  :- parent( X, Y).
└──голова──┘        └──тело──┘
```

Если условие parent(X, Y) является истинным, то его логическим следствием становится offspring(Y, X).

Применение правил в языке Prolog иллюстрируется в следующем примере. Зададим программе вопрос, является ли Лиз дочерью Тома:

```
?- offspring( liz, tom).
```

Поскольку в программе отсутствуют факты о дочерях и сыновьях, единственным способом поиска ответа на этот вопрос является использование правила с определением отношения offspring. Данное правило является общим в том смысле, что оно применимо к любым объектам X и Y; но его можно также применить и к таким конкретным объектам, как liz и tom. Чтобы применить правило к объектам liz и tom, вместо Y необходимо подставить liz, а вместо X — tom. Это действие называется *конкретизацией переменных* (в данном случае — X и Y), которое выполняется следующим образом:

```
X = tom и Y = liz
```

После конкретизации будет получен частный случай общего правила, который выглядит следующим образом:

```
offspring( liz, tom) :- parent( tom, liz).
```

Часть с обозначением условия принимает вид

```
parent( tom, liz)
```

После этого система Prolog пытается определить, является ли истинной часть с обозначением условия. Поэтому первоначальная цель:

```
offspring( liz, tom)
```

заменяется подцелью:

```
parent( tom, liz)
```

Оказалось, что задача достижения этой (новой) цели является тривиальной, поскольку ее можно найти как факт в рассматриваемой программе. Это означает, что часть данного правила с обозначением заключения также является истинной, и Prolog в качестве ответа на вопрос выводит `yes`.

Теперь введем в рассматриваемый пример программы еще некоторую информацию о семейных отношениях. Определение отношения `mother` может быть основано на следующем логическом утверждении:

```
Для всех X и Y,
  X является матерью Y, если
  X является одним из родителей Y и
  X - женщина.
```

Это утверждение можно перевести на язык Prolog в виде следующего правила:

```
mother( X, Y) :- parent( X, Y), female(X).
```

Запятая между двумя условиями указывает на конъюнкцию этих условий; это означает, что оба условия должны быть истинными.

Такие отношения, как `parent`, `offspring` и `mother`, можно проиллюстрировать с помощью схем, подобных приведенным на рис. 1.3. Эти схемы соответствуют следующим соглашениям. Узлы графов относятся к объектам, т.е. параметрам отношений. Дуги между узлами соответствуют бинарным (или двухместным) отношениям. Дуги направлены от первого параметра отношения ко второму. Унарные отношения обозначаются на схемах путем проставления отметки на соответствующих объектах с именем отношения. Отношения, которые определены на основе других отношений, представлены в виде пунктирных дуг. Поэтому каждую схему необходимо интерпретировать следующим образом: если соблюдаются отношения, обозначенные сплошными дугами, то соблюдаются и созданные на их основе отношения, обозначенные пунктирными дугами. Согласно рис. 1.3, отношение `grandparent` можно непосредственно записать на языке Prolog следующим образом:

```
grandparent( X, Z) :- parent( X, Y), parent( Y, Z).
```

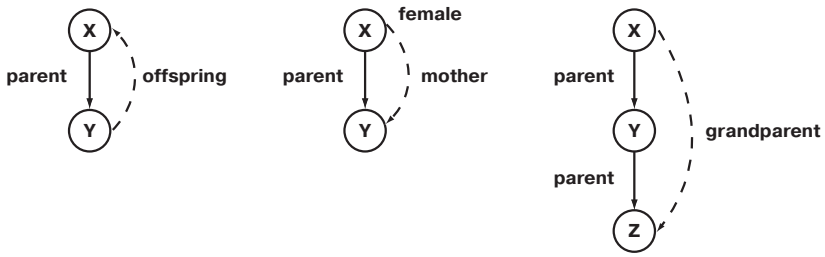

Рис. 1.3. Графы, которые определяют отношения `offspring`, `mother` и `grandparent` в терминах других отношений

На данном этапе необходимо кратко рассмотреть вопрос о компоновке программ. Система Prolog предоставляет почти полную свободу выбора компоновки программ. Поэтому программист может вставлять в текст программы пробелы и пустые строки в полном соответствии со своими вкусами. Но, как правило, следует стремиться к тому, чтобы программы выглядели четкими и аккуратными и, самое главное, были удобными для чтения. Для этого чаще всего голова предложения и каждая цель в его теле записываются на отдельной строке. При этом желательно обозначать цели отступом, чтобы различия между головой и целями стали более очевидными. Например, в соответствии с этими соглашениями правило `grandparent` должно быть записано следующим образом:

```
grandparent( X, Z) :-
  parent( X, Y),
  parent( Y, Z).
```

Схема отношения `sister` (рис. 1.4) имеет следующее определение:

```
Для любого X и Y
  X является сестрой Y, если
  1) X и Y имеют общего родителя и
  2) X - женщина.
```

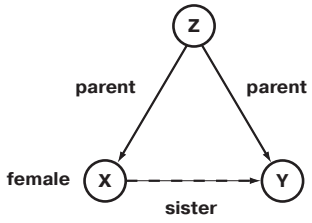

Рис. 1.4. Определение отношения `sister`

Граф, представленный на рис. 1.4, можно перевести на язык Prolog следующим образом:

```
sister( X, Y) :- parent( Z, X), parent( Z, Y), female(X).
```

Обратите внимание на то, каким способом было выражено требование "`X и Y имеют общего родителя`". Для этого использовалась следующая логическая формулировка: `некоторый Z должен быть родителем X, и тот же Z должен быть родителем Y`. Альтернативный, но менее изящный способ мог предусматривать использование следующей цепочки утверждений: `Z1 является родителем X, Z2 является родителем Y и Z1 равно Z2`.

Теперь системе можно задать вопрос:

```
?- sister( ann, pat).
```

Ответом должно быть "`yes`", как и следовало ожидать (см. рис. 1.1). Поэтому можно сделать вывод, что отношение `sister` в том виде, в каком оно определено, действует правильно. Но в рассматриваемой программе имеется незаметный на первый взгляд недостаток, который обнаруживается при получении ответа на вопрос о том, кто является сестрой Пэт:

```
?- sister( X, pat).
```

Prolog находит два ответа, и один из них может оказаться неожиданным.

```
X = ann;
X = pat
```

Итак, Пэт является сестрой самой себя?! По-видимому, такой исход не подразумевался при определении отношения `sister`. Но согласно правилу, касающемуся сестер, ответ системы Prolog является полностью обоснованным. В правиле о сестрах нет упоминания о том, что `X` и `Y` не должны быть одинаковыми, если `X` рассматривается как сестра `Y`. Поскольку это требование не предъявляется, система Prolog (вполне обоснованно) предполагает, что `X` и `Y` могут быть одинаковыми, и поэтому приходит к заключению, что любая женщина, имеющая родителя, является сестрой самой себя.

Чтобы исправить приведенное выше правило о сестрах, необходимо дополнительно указать, что `X` и `Y` должны быть разными. Как показано в следующих главах, такую задачу можно решить несколькими способами, но на данный момент предположим, что системе Prolog уже известно отношение `different` и условие

```
different( X, Y)
```

удовлетворяется, если и только если X и Y не равны. Поэтому усовершенствованное правило для отношения sister может выглядеть следующим образом:

```
sister( X, Y) :-
  parent( Z, X),
  parent( Z, Y),
  female( X),
  different( X, Y).
```

На основании изложенного в этом разделе можно сделать следующие важные выводы.

- Программы Prolog можно дополнять, вводя новые предложения.
- Предложения Prolog относятся к трем типам: факты, правила и вопросы.
- С помощью фактов можно вводить в программу сведения, которые всегда и безусловно являются истинными.
- С помощью правил можно вводить в программу сведения, которые являются истинными в зависимости от заданного условия.
- Задавая программе вопросы, пользователь может узнавать, какие сведения являются истинными.
- Предложения языка Prolog состоят из головы и тела. Тело представляет собой список целей, разделенных запятыми. Запятые рассматриваются как знаки конъюнкции.
- Факты представляют собой предложения, которые имеют голову и пустое тело. Вопросы имеют только тело. Правила имеют голову и (непустое) тело.
- В процессе вычисления переменные можно заменять другими объектами. В таком случае переменная становится конкретизированной.
- Предполагается, что на переменные распространяется действие квантора всеобщности, который имеет словесное выражение “для всех”. Но если переменные появляются только в теле, их можно трактовать несколькими способами. Например, предложение

  ```
  hasachild( X) :- parent( X, Y).
  ```

  можно прочитать двумя приведенными ниже способами.

  ```
  а) Для всех X и Y,
       если X является родителем Y, то
       X имеет ребенка.
  б) Для всех X,
       X имеет ребенка, если
       существует некоторый Y, такой, что X является родителем Y.
  ```

Упражнения

1.3. Преобразуйте приведенные ниже утверждения в правила Prolog.

а) Каждый, кто имеет ребенка, счастлив (введите отношение happy с одним параметром).

б) Для всех X, если X имеет ребенка, имеющего сестру, то X имеет двоих детей (введите новое отношение hastwochildren).

1.4. Определите отношение grandchild с использованием отношения parent. Подсказка: оно должно быть аналогично отношению grandparent (см. рис. 1.3).

1.5. Определите отношение aunt(X, Y) в терминах отношений parent и sister. Для упрощения этой задачи вы можете вначале нарисовать схему для отношения, определяющего понятие aunt (тетя), в стиле рис. 1.3.

1.3. Рекурсивные правила

Введем еще одно отношение в рассматриваемую программу с описанием семьи — отношение `predecessor` (предок). Это отношение будет определено в терминах отношения `parent`. Его можно представить с помощью двух правил. Первое правило определяет прямых (непосредственных) предков, а второе правило — непрямых предков. Говорят, что некоторый `X` является непрямым предком некоторого `Z`, если существует цепочка родительских связей между людьми от `X` до `Z`, как показано на рис. 1.5. В примере, приведенном на рис. 1.1, Том является прямым предком Лиз и непрямым предком Пэт.

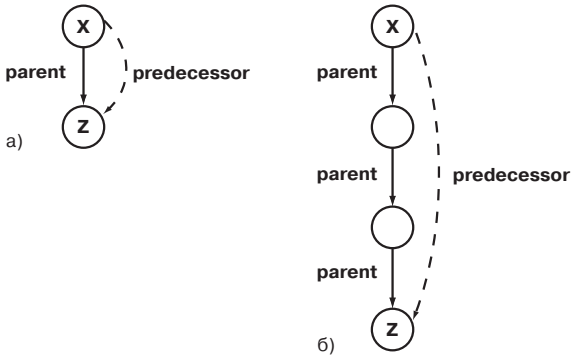

Рис. 1.5. Примеры отношения `predecessor`: а) `X` является прямым предком `Z`; б) `X` является непрямым предком `Z`

Первое правило является простым и может быть сформулировано следующим образом:

```
Для всех X и Z,
  X - предок Z, если
  X - родитель Z.
```

Это утверждение можно сразу же перевести на язык Prolog таким образом:

```
predecessor( X, Z) :-
  parent( X, Z).
```

Второе правило, с другой стороны, является более сложным, поскольку решение задачи представления цепочки родительских связей может вызвать некоторые проблемы. Одна из попыток определить непрямых предков показана на рис. 1.6. Согласно этому рисунку, отношение `predecessor` должно быть определено как множество следующих предложений:

```
predecessor( X, Z) :-
  parent( X, Z).

predecessor( X, Z) :-
  parent( X, Y),
  parent( Y, Z).

predecessor( X, Z) :-
  parent( X, Y1),
  parent( Y1, Y2),
  parent( Y2, Z).

predecessor( X, Z) :-
  parent( X, Y1),
```

```
  parent( Y1, Y2),
  parent( Y2, Y3),
  parent( Y3, Z).
...
```

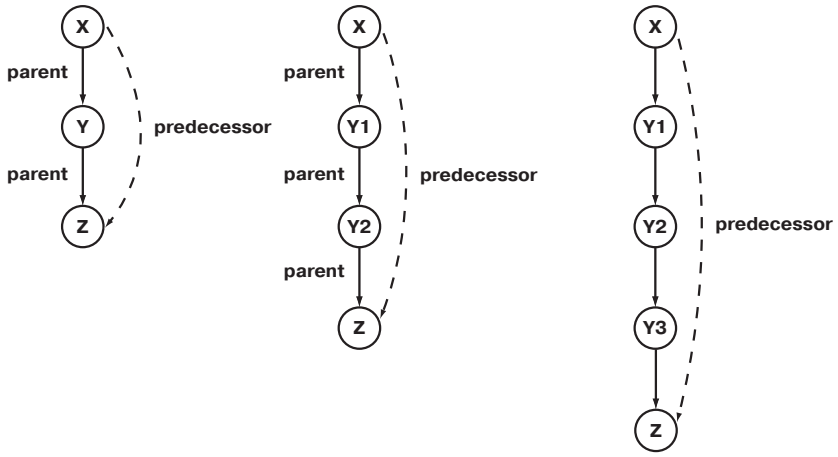

Рис. 1.6. Пары предков и потомков, находящихся друг от друга на разных расстояниях

Эта программа является слишком длинной, но еще более важно то, что пределы ее действия довольно ограничены. Она позволяет находить предков в генеалогическом дереве только до определенного уровня, поскольку длина цепочки людей между предками и потомками лимитируется длиной существующих предложений с определением предков.

Но для формулировки отношения `predecessor` можно применить гораздо более изящную и правильную конструкцию. Она является правильной в том смысле, что позволяет находить предков до любого колена. Основная идея состоит в том, что отношение `predecessor` должно быть определено в терминах себя самого. Эта идея иллюстрируется на рис. 1.7 и выражается в виде следующего логического утверждения:

```
Для всех X и Z,
  X - предок Z,
  если имеется такой Y, что
  1) X - родитель Y и
  2) Y - предок Z.
```

Предложение Prolog, имеющее такой же смысл, приведено ниже.

```
predecessor( X, Z) :-
  parent( X, Y),
  predecessor( Y, Z).
```

Таким образом, сформулирована полная программа для отношения `predecessor`, которая состоит из двух правил: первое из них определяет прямых, а вторая — непрямых предков. Оба правила, записанные вместе, приведены ниже.

```
predecessor( X, Z) :-
  parent( X, Z).

predecessor( X, Z) :-
  parent( X, Y),
  predecessor( Y, Z).
```

Ключом к анализу этой формулировки является то, что отношение `predecessor` используется для определения самого себя. Такая конструкция может показаться на

первый взгляд непонятной, поскольку возникает вопрос, можно ли определить некоторое понятие, используя для этого то утверждение, которое само еще не было полностью определено. Подобные определения имеют общее название *рекурсивных*. С точки зрения логики они являются полностью правильными и понятными; кроме того, они становятся очевидными после изучения схемы, приведенной на рис. 1.7. Но остается нерешенным вопрос, способна ли система Prolog использовать рекурсивные правила. Как оказалась, эта система действительно способна очень легко применять рекурсивные определения. Рекурсивное программирование фактически является одним из фундаментальных принципов программирования на языке Prolog. Без использования рекурсии на языке Prolog невозможно решать какие-либо сложные задачи.

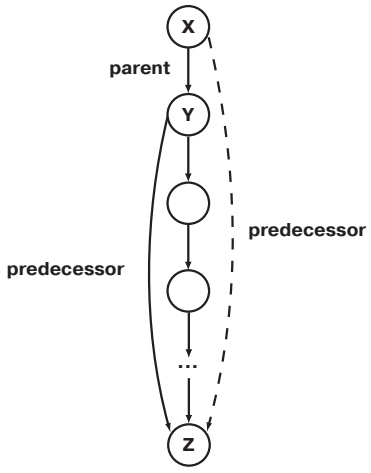

Рис. 1.7. Рекурсивная формулировка отношения predecessor

Возвращаясь к рассматриваемой программе, можно задать системе Prolog вопрос о том, кто является потомками Пэм. Иными словами, для кого Пэм является предком?

```
?- predecessor( pam, X).
X = bob;
X = ann;
X = pat;
X = jim
```

Ответы системы Prolog, безусловно, верны и логически следуют из определения отношений `predecessor` и `parent`. Тем не менее остается открытым весьма важный вопрос: как фактически система Prolog использует программу для поиска этих ответов?

Неформальное описание того, как эта задача решается в системе Prolog, приведено в следующем разделе. Но вначале соединим все фрагменты программы с описанием семьи, которая постепенно совершенствовалась путем добавления новых фактов и правил. Окончательная форма этой программы приведена в листинге 1.1. Прежде чем перейти к описанию этого листинга, необходимо сделать следующие замечания: во-первых, дать определение термина *процедура*, а во-вторых, отметить, как используются комментарии в программах.

Листинг 1.1. Программа `family` с описанием семьи

```
parent( pam, bob).      % Пэм является одним из родителей Боба
parent( tom, bob).
parent( tom, liz).
parent( bob, ann).
parent( bob, pat).
```

```
parent( pat, jim).

female( pam).           % Пэм - женщина
male( tom).             % Том - мужчина
male( bob).
female( liz).
female( ann).
female( pat).
male( jim).

offspring( Y, X) :-     % Y является сыном или дочерью X, если
  parent( X, Y).        % X является одним из родителей Y

mother( X, Y) :-        % X является матерью Y, если
  parent( X, Y),        % X является одним из родителей Y и
  female( X).           % X - женщина

grandparent( X, Z) :-   % X является дедушкой или бабушкой Z, если
  parent( X, Y),        % X является одним из родителей Y и
  parent( Y, Z).        % Y является одним из родителей Z

sister( X, Y) :-        % X является сестрой Y, если
  parent( Z, X),
  parent( Z, Y),        % X и Y имеют одного и того же родителя и
  female( X),           % X - женщина и
  different( X, Y).     % X и Y являются разными

predecessor( X, Z) :-   % Правило pr1: X - предок Z
  parent( X, Z).

predecessor( X, Z) :-   % Правило pr2: X - предок Z
  parent( X, Y),
  predecessor( Y, Z)
```

В программе, приведенной в листинге 1.1, определено несколько отношений: `parent`, `male`, `female`, `predecessor` и т.д. Например, отношение `predecessor` определено с помощью двух предложений. Принято считать, что оба эти предложения касаются отношения `predecessor`. Иногда удобно рассматривать как единое целое все множество предложений, касающихся одного и того же отношения. Подобный набор предложений называется *процедурой*.

В листинге 1.1 два правила, касающиеся отношения `predecessor`, обозначены разными именами, `pr1` и `pr2`, которые указаны в виде комментариев к программе. Эти имена будут использоваться в дальнейшем в качестве ссылок на данные правила. Обычно комментарии игнорируются системой Prolog. Они служат лишь в качестве дополнительно пояснения для лица, читающего программу. Комментарии в языке Prolog отделяются от остальной части программы специальными символьными скобками "`/*`" и "`*/`". Таким образом, комментарии в языке Prolog выглядят следующим образом:

```
/* Это - комментарий */
```

Еще один метод, более удобный для оформления коротких комментариев, предусматривает использование знака процента "`%`". Все, что находится между знаком "`%`" и концом строки, интерпретируется как комментарий.

```
% Это - также комментарий
```

Упражнение

1.6. Рассмотрим следующее альтернативное определение отношения `predecessor`:

```
predecessor( X, Z) :-
  parent( X, Z).
```

```
predecessor( X, Z) :-
  parent( Y, Z),
  predecessor( X, Y).
```

Можно ли это определение отношения `predecessor` также считать правильным? Откорректируйте схему, приведенную на рис. 1.7, чтобы она соответствовала этому новому определению.

1.4. Общие принципы поиска ответов на вопросы системой Prolog

В этом разделе дано неформальное описание процесса поиска ответов на вопросы в системе Prolog. Вопрос к системе Prolog всегда представляет собой последовательность из одной или нескольких целей. Чтобы ответить на вопрос, Prolog пытается достичь всех целей. Но что в данном контексте означает выражение "достичь цели"? Достичь цели — это значит продемонстрировать, что цель является истинной, при условии, что отношения в программе являются истинными. Другими словами, выражение *достичь цели* означает: продемонстрировать, что цель логически следует из фактов и правил, заданных в программе. Если вопрос содержит переменные, система Prolog должна также найти конкретные объекты (вместо переменных), при использовании которых цели достигаются. Для пользователя отображаются варианты конкретизации переменных, полученные при подстановке конкретных объектов вместо переменных. Если система Prolog не может продемонстрировать для какого-то варианта конкретизации переменных, что цели логически следуют из программы, то выдает в качестве ответа на вопрос слово "`no`".

Таким образом, с точки зрения математики программу Prolog следует интерпретировать так: Prolog принимает факты и правила как набор аксиом, а вопрос пользователя — как теорему, требующую доказательства; затем Prolog пытается доказать теорему, т.е. продемонстрировать, что она является логическим следствием из аксиом.

Проиллюстрируем этот подход к описанию работы системы Prolog на классическом примере. Предположим, что заданы приведенные ниже аксиомы.

```
Все люди способны ошибаться.
Сократ - человек.
```

Из этих двух аксиом логически следует теорема:

```
Сократ способен ошибаться.
```

Первая аксиома, приведенная выше, может быть переформулирована следующим образом:

```
Для всех X, если X - человек, то X способен ошибаться.
```

Соответствующим образом этот пример может быть переведен на язык Prolog, как показано ниже.

```
fallible( X) :- man( X).   % Все люди способны ошибаться
man( socrates).            % Сократ - человек
?- fallible( socrates).    % Сократ способен ошибаться?
yes                        % Да
```

Более сложный пример, взятый из программы с описанием семьи (см. листинг 1.1), представлен ниже.

```
?- predecessor( tom, pat).
```

Известно, что в этой программе определен факт `parent( bob, pat)`. Используя этот факт и правило `pr1`, можно прийти к заключению, что имеет место факт `predecessor( bob, pat)`. Это — производный факт, в том смысле, что его нельзя непосредственно найти в программе, но можно вывести из фактов и правил программы. Этап вывода, подобный этому, можно записать в более компактной форме следующим образом:

```
parent( bob, pat) ==> predecessor( bob, pat)
```

Это выражение можно прочесть так: из факта `parent( bob, pat)` следует факт `predecessor( bob, pat)`, согласно правилу `pr1`. Кроме того, известно, что определен факт `parent( tom, bob)`. Используя этот факт и производный факт `predecessor( bob, pat)`, можно сделать вывод, что имеет место факт `predecessor( tom, pat)`, согласно правилу `pr2`. Тем самым показано, что целевое выражение `predecessor( tom, pat)` является истинным. Весь этот двухэтапный процесс вывода можно записать следующим образом:

```
parent( bob, pat) ==> predecessor( bob, pat)
parent( tom, bob) и predecessor( bob, pat) ==> predecessor( tom, pat)
```

Итак, было показано, что может существовать последовательность шагов, позволяющих достичь определенной цели, т.е. выяснить, что цель является истинной. Такая последовательность шагов называется *последовательностью доказательства*. Тем не менее еще не показано, как фактически система Prolog находит такую последовательность доказательства.

Prolog ищет последовательность доказательства в порядке, обратном тому, который был только что использован. Эта система начинает не с простых фактов, заданных в программе, а с целей, и с помощью правил заменяет текущие цели новыми до тех пор, пока не обнаружится, что новые цели являются простыми фактами. Рассмотрим вопрос:

```
?- predecessor( tom, pat).
```

Система Prolog пытается достичь данной цели. Для этого она предпринимает попытки найти в программе предложение, из которого непосредственно может следовать указанная выше цель. Очевидно, что в этом случае подходящими являются только предложения `pr1` и `pr2`. Это — правила, касающиеся отношения `predecessor`. Говорят, что головы этих правил соответствуют цели.

Два предложения, `pr1` и `pr2`, представляют собой два альтернативных способа дальнейших действий системы Prolog. Она вначале проверяет предложение, которая находится на первом месте в программе:

```
predecessor( X, Z) :- parent( X, Z).
```

Поскольку целью является факт `predecessor( tom, pat)`, необходимо выполнить конкретизацию переменных в этом правиле следующим образом:

```
X = tom, Z = pat
```

Затем первоначальная цель `predecessor( tom, pat)` заменяется следующей новой целью:

```
parent( tom, pat)
```

Данный этап использования правила преобразования цели в другую цель, как указано выше, схематически показан на рис. 1.8. В программе отсутствует предложение, которое соответствует цели `parent( tom, pat)`, поэтому такая цель непосредственно не достижима. Теперь система Prolog возвращается к первоначальной цели, чтобы проверить альтернативный способ вывода главной цели `predecessor( tom, pat)`. Поэтому правило `pr2` проверяется следующим образом:

```
predecessor( X, Z) :-
  parent( X, Y),
  predecessor( Y, Z).
```

Как было указано выше, конкретизация переменных `X` и `Z` выполняется следующим образом:

```
X = tom, Z = pat
```

Но конкретизация `Y` еще не выполнена. Верхняя цель `predecessor( tom, pat)` заменяется двумя следующими целями:

```
parent( tom, Y),
predecessor( Y, pat)
```

Этот этап выполнения показан на рис. 1.9, который демонстрирует дальнейшее развитие ситуации, представленной на рис. 1.8.

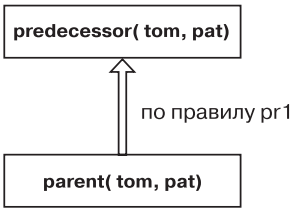

Рис. 1.8. Первый этап выполнения программы; цель, показанная вверху, является истинной, если истинна цель, показанная внизу

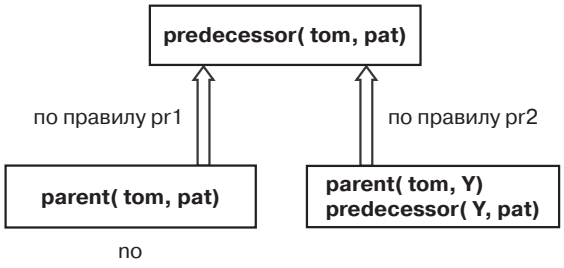

Рис. 1.9. Продолжение схемы выполнения программы, показанной на рис. 1.8

Теперь система Prolog сталкивается с необходимостью заниматься двумя целями и пытается их достичь в том порядке, в каком они записаны. Первая цель достигается легко, поскольку она соответствует одному из фактов в программе. Это соответствие вынуждает выполнить конкретизацию переменной `Y` и подстановку вместо нее значения `bob`. Таким образом, достигается первая цель, и оставшаяся цель принимает вид

```
predecessor( bob, pat)
```

Для достижения данной цели снова используется правило `pr1`. Следует отметить, что это (второе) применение того же правила не имеет ничего общего с его предыдущим применением. Поэтому система Prolog использует в правиле новый набор переменных при каждом его применении. Чтобы продемонстрировать это, переименуем переменные в правиле `pr1` для этого этапа применения правила следующим образом:

```
predecessor( X', Z') :-
  parent( X', Z').
```

Голова данного правила должна соответствовать текущей цели `predecessor ( bob, pat)`, поэтому:

```
X' = bob,
Z' = pat
```

Текущая цель заменяется следующей:

```
parent( bob, pat)
```

Данная цель достигается сразу же, поскольку она представлена в программе в виде факта. На этом завершается формирование схемы выполнения, которая представлена в графическом виде на рис. 1.10.

Графическая схема выполнения программы (см. рис. 1.10) имеет форму дерева. Узлы дерева соответствуют целям или спискам целей, которые должны быть достигнуты. Дуги между узлами соответствуют этапам применения (альтернативных) предложений программы, на которых цели одного узла преобразуются в цели другого узла. Верхняя цель достигается после того, как будет найден путь от корневого узла (верхней цели) к лист-узлу, обозначенному как "`yes`". Лист носит метку "`yes`", если он представляет собой простой факт. Процесс выполнения программ Prolog состоит в поиске путей, оканчивающихся такими простыми фактами. В ходе поиска система Prolog может войти в одну из ветвей, не позволяющих достичь успеха. При обнаружении того, что ветвь не позволяет достичь цели, система Prolog автоматически возвращается к предыдущему узлу и пытается использовать в этом узле альтернативное предложение.

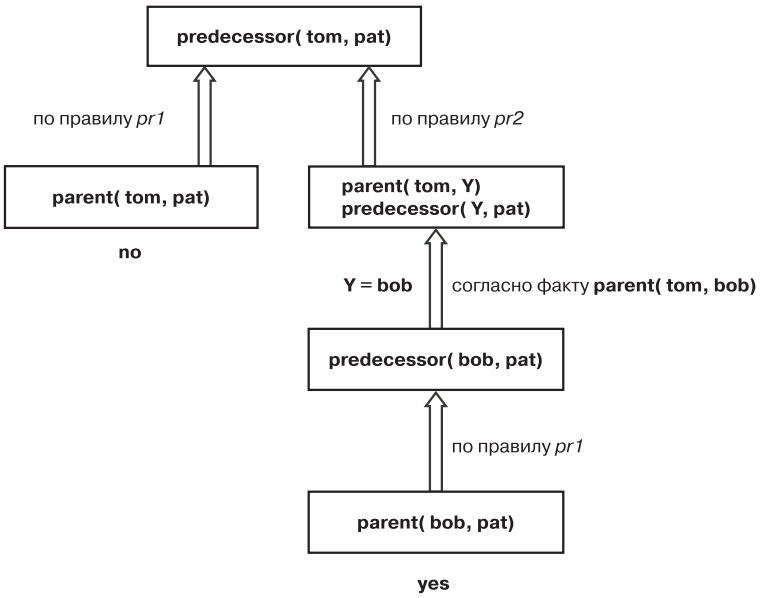

Рис. 1.10. Полная схема процесса достижения цели `predecessor( tom, pat)`*. Правая ветвь показывает, что цель является достижимой*

Упражнение

1.7. Укажите, каким образом система Prolog вырабатывает ответы на следующие вопросы с использованием программы, приведенной в листинге 1.1. Составьте соответствующие схемы выполнения в стиле рис. 1.8–1.10. Происходят ли возвраты при поиске ответов на некоторые вопросы?

а) `?- parent( pam, bob).`

б) `?- mother( pam, bob).`

в) `?- grandparent( pam, ann).`

г) `?- grandparent( bob, jim).`

1.5. Декларативное и процедурное значение программ

В рассмотренных выше примерах всегда было возможно понять результаты программы, не зная точно, как система фактически нашла эти результаты. Но иногда важно учитывать, как именно происходит поиск ответа в системе, поэтому имеет смысл проводить различие между двумя уровнями значения программ Prolog, а именно, между

- декларативным значением и
- процедурным значением.

Декларативное значение касается только отношений, определенных в программе. Поэтому декларативное значение регламентирует то, каким должен быть результат работы программы. С другой стороны, процедурное значение определяет также способ получения этого результата, иными словами, показывает, как фактически проводится обработка этих отношений системой Prolog.

Способность системы Prolog самостоятельно отрабатывать многие процедурные детали считается одним из ее особых преимуществ. Prolog побуждает программиста в первую очередь рассматривать декларативное значение программ, в основном независимо от их процедурного значения. А поскольку результаты программы в принципе определяются их декларативным значением, этого должно быть (по сути) достаточно для написания программы. Такая особенность важна и с точки зрения практики, так как декларативные аспекты программ обычно проще для понимания по сравнению с процедурными деталями. Но чтобы полностью воспользоваться этими преимуществами, программист должен сосредоточиваться в основном на декларативном значении и, насколько это возможно, избегать необходимости отвлекаться на детали выполнения. Последние необходимо оставлять в максимально возможной степени для самой системы Prolog.

Такой декларативный подход фактически часто позволяет гораздо проще составлять программы на языке Prolog по сравнению с такими типичными процедурно-ориентированными языками программирования, как C или Pascal. Но, к сожалению, декларативный подход не всегда позволяет решить все задачи. По мере дальнейшего изучения материала этой книги станет очевидно, что процедурные аспекты не могут полностью игнорироваться программистом по практическим причинам, связанным с обеспечением вычислительной эффективности, особенно при разработке больших программ. Тем не менее необходимо стимулировать декларативный стиль мышления при разработке программ Prolog и игнорировать процедурные аспекты до такой степени, которая является допустимой с точки зрения практических ограничений.

Резюме

- *Программирование на языке Prolog* представляет собой процесс определения отношений и выдачи системе запросов об этих отношениях.
- Программа состоит из *предложений*, которые относятся к трем типам: факты, правила и вопросы.
- *Отношение* может быть определено на основе фактов путем задания n-элементных кортежей объектов, которые удовлетворяют отношению, или путем задания правил, касающихся этого отношения.
- *Процедура* представляет собой набор предложений, касающихся одного и того же отношения.
- *Выполнение запросов* об отношениях, передаваемых системе в виде вопросов, напоминает выполнение запросов к базе данных. Ответ системы Prolog на вопрос состоит из множества объектов, которые соответствуют этому вопросу.
- В системе Prolog для определения того, соответствует ли объект вопросу, часто применяется сложный процесс, который связан с выполнением логического вывода, рассмотрением многих альтернатив и, возможно, *перебора с возвратами*. Все эти операции выполняются системой Prolog автоматически и, в принципе, скрыты от пользователя.
- Различаются два *значения программ* Prolog: декларативное и процедурное. Декларативный подход является более привлекательным с точки зрения программирования. Тем не менее программисту также часто приходится учитывать процедурные детали.
- В данной главе представлены следующие понятия:
  - предложение, факт, правило, вопрос;
  - голова предложения, тело предложения;
  - рекурсивное правило, рекурсивное определение;
  - процедура;

- атом, переменная;
- конкретизация переменной;
- цель;
- цель, которая является достижимой, цель, которая достигнута;
- цель, которая является недостижимой, цель, которая не достигнута;
- перебор с возвратами;
- декларативное значение, процедурное значение.

Дополнительные источники информации

В различных реализациях Prolog используются разные синтаксические соглашения. Но большинство из них следует традициям так называемого *эдинбургского синтаксиса* (известного также как синтаксис DEC-10, сложившийся под влиянием реализации Prolog для компьютера DEC-10, которая оставила заметный след в истории развития этого языка [9], [122]). Кроме того, эдинбургский синтаксис лежит в основе международного стандарта ISO языка Prolog, ISO/IEC 13211-1 [43]. В настоящее время основные реализации Prolog главным образом совместимы с этим стандартом. В данной книге используется подмножество стандартного синтаксиса, если не считать некоторых небольших и незначительных различий. В тех редких случаях, когда допускаются такие различия, в соответствующем месте книги дается примечание на этот счет.