Лекция 9. Логическое программирование. Правила и запросы

Ключевые понятия: ***правило***, ***рекурсивное правило, конкретизация переменной, достижимая цель, недостижимая цель, перебор с возвратами, декларативная программа, процедурная программа***

1 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.

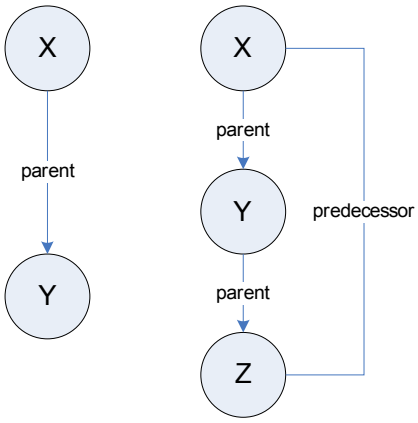


Рис.1

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

predecessor (X, Z) :- parent(X, Y), parent(Y, Z).
predecessor (X, Z) :- parent(X, Y1), ... parent(YN-1, YN), parent(YN, Z)

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.2 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.

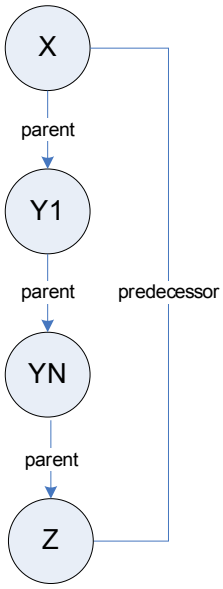
Предложение Prolog, имеющее такой же смысл, приведено ниже.

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. 2. 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.

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 = ami; 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 далее. 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. 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.

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

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

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


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

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

sisterf 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 рг2: X - предок Z
  parent( X Y),
  predecessor( Y, Z)
```

В программе, приведенной в листинге 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 два правила, касающиеся отношения predecessor, обозначены разными именами, prl и рг2, 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.
```

2 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), представлен ниже. ?- predecessor(tom, pat).

Известно, что в этой программе определен факт parent(bob, pat). Используя этот факт и правило prl, 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 prl. Кроме того, известно, что определен факт parent(torn, bob). Используя этот факт и производный факт predecessor(bob, pat), можно сделать вывод, что имеет место факт predecessor(torn, pat) , согласно правилу рг2. Тем самым показано, что целевое выражение predecessor(torn, pat)

является истинным. Весь этот двухэтапный процесс вывода можно записать следующим образом:

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

parent(torn, bob) & predecessor(bob, pat) ==> predecessor(torn, 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 prl и pr2. Это - правила, касающиеся отношения predecessor. Говорят, что головы этих правил соответствуют цели.

Два предложения, prl 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(torn, pat), необходимо выполнить конкретизацию переменных в этом правиле следующим образом:

X = tom, Z = pat

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

Данный этап использования правила преобразования цели в другую цель, как указано выше, схематически показан на рис. 3. 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(torn, 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).

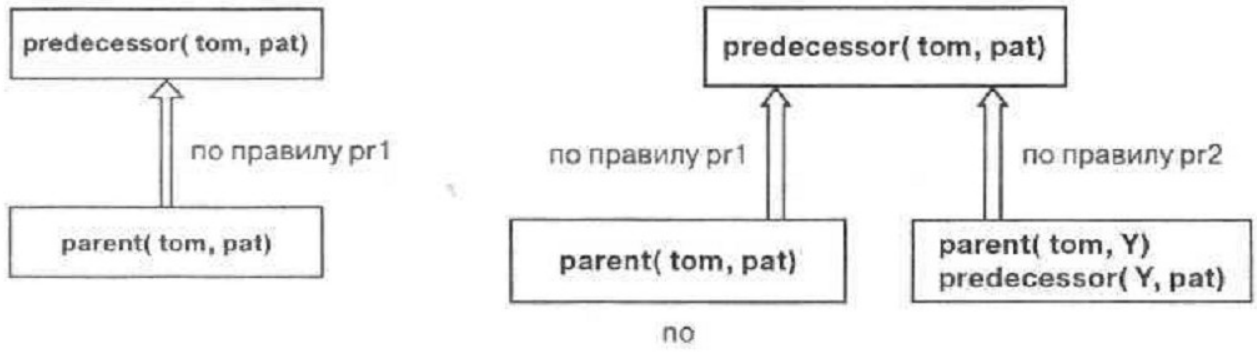


Рис.3

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

predecessort (bob, pat)

Для достижения данной цели снова используется правило prl. 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 prl 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. 4.

Графическая схема выполнения программы (см. рис. 4) 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.

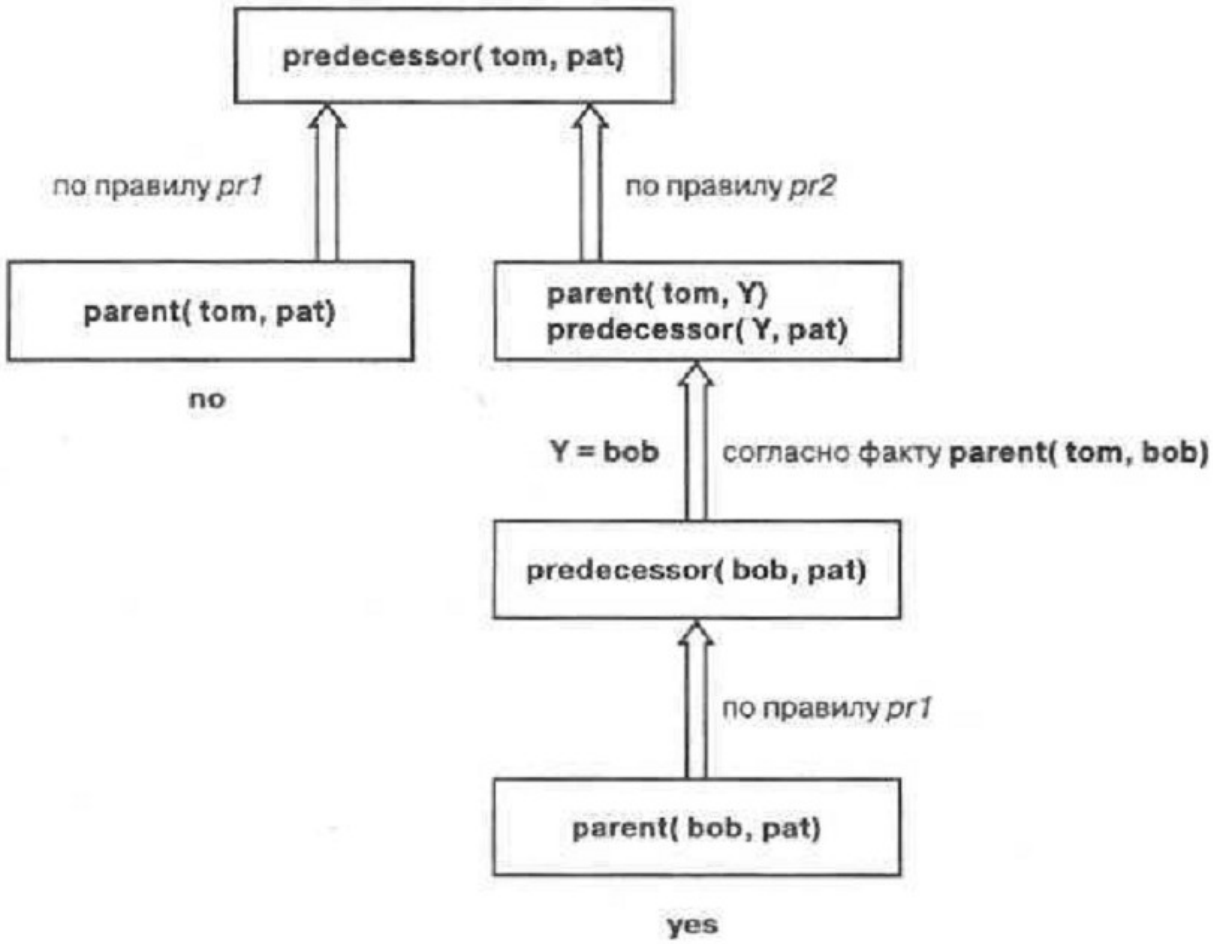


Рис.4

3 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 программировании, как C или Pascal.

4 Резюме

- Программирование па 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.