Лекция 11. Логическое программирование. Согласование

Ключевые понятия: ***согласование***, **.**

1 Согласование

В предыдущем разделе было показано, как могут использоваться термы для представления сложных объектов данных. Наиболее важной операцией с термами является согласование. Даже одно только согласование позволяет выполнять некоторые интересные вычисления.

Если даны два терма, то можно утверждать, что они согласуются, если выполнены следующие условия.

1 Они являются идентичными.

2. Переменные в обоих термах можно конкретизировать значениями объектов таким образом, чтобы после подстановки этих объектов вместо переменных термы стали идентичными,

Например, термы date(D, M, 2001) и date(D1, may, Y1) согласуются. Ниже показана одна конкретизация, при которой оба терма становятся идентичными.

• D конкретизируется значением D1.

• M конкретизируется значением may.

• Y1 конкретизируется значением 2001.

Этот вариант конкретизации можно записать более компактно в знакомой форме, в которой Prolog выводит результаты:

D = D1

M = may

Y1 = 2001

С другой'стороны, термы date(D, M, 2001) и date(D1, M1, 1444) не согласуются; то же самое касается термов date(X, Y, *Z)* и point (X, Y, Z),

Согласованием называется процесс, в котором в качестве входных данных берутся два терма и выполняется проверка того, являются ли они согласованными. Если термы не согласуются, это означает, что процесс согласования оканчивается неудачей, а если они согласуются, то этот процесс завершается успешно и в нем осуществляется также конкретизация переменных в обоих термах такими значениями, что оба терма становятся идентичными.

Снова рассмотрим пример согласования двух дат. Запрос на выполнение

этой операции можно передать системе Prolog в виде следующего вопроса с использованием знака операции "=":

?- date (D, M, 2001) = date(D1, may, Yl) .

Выше уже упоминалась конкретизация D = D1, M = may, Yl = 2001, при которой достигается согласование. Но есть и другие варианты конкретизации, при которых оба терма становятся идентичными. Два из них приведены ниже.

D = 1 D1 = 1 M = may Yl = 2001

D = third Dl = third M = may Y1 = 2001

Принято считать, что эти две конкретизации являются менее общими по сравнению с первой, поскольку они ограничивают значения переменных D и D1 более жестко, чем это необходимо. Для того чтобы оба терма в данном примере стали идентичными, требуется лишь предусмотреть применение одинаковых значений D и D1, хотя в качестве этого значения можно использовать что угодно. Согласование в языке Prolog всегда приводит к *наиболее общей конкретизации.* Таковой является конкретизация, которая ограничивает переменные в минимально возможной степени и тем самым оставляет максимально возможную свободу для дальнейшей конкретизации (на тот случай, если потребуется дальнейшее согласование). В качестве примера рассмотрим следующий вопрос:

?- date(D, M, 2001) = date (Dl, may, Yl) ,

date (D, M, 2001) = date (15, M, Y).

Для достижения первой цели система Prolog конкретизирует переменные следующим образом:

D = D1 M = may Y1 = 2001

После достижения второй цели конкретизация становится более определенной, как показано ниже.

D = 15 D1 = 15

M = may Yl = 2001 Y = 2001

Этот пример также показывает, что в ходе последовательного достижения цели переменные обычно конкретизируются все более определенными значениями. Ниже приведены правила определения того, согласуются ли два терма, S и T.

1. Если S и T являются константами, то они согласуются, только если представляют собой одинаковый объект.

2. Если S представляет собой переменную, а T - нечто иное, то они согласуются и S конкретизируется значением T. И наоборот, если переменной является T, то T конкретизируется значением S.

3. Если S и T являются структурами, то они согласуются, только если выполняются следующие условия:

а) S и T имеют одинаковый главный функтор;

б) все их соответствующие компоненты согласуются.

Результирующая конкретизация определяется путем согласования компонентов.

Последнее из этих правил можно проиллюстрировать графически, рассматривая древовидное представление термов, как в примере, показанном на рис. 7. Процесс согласования начинается с корня (с главных функторов). Если оба функтора согласуются, процесс переходит к параметрам и осуществляется согласование пар соответствующих параметров. Итак, весь процесс согласования можно рассматривать как состоящий из показанной ниже последовательности (простых) операций согласования.

triangle = triangle,

point(1,1) = X,

A = point(4,Y),

point (2,3) = point(2,Z)

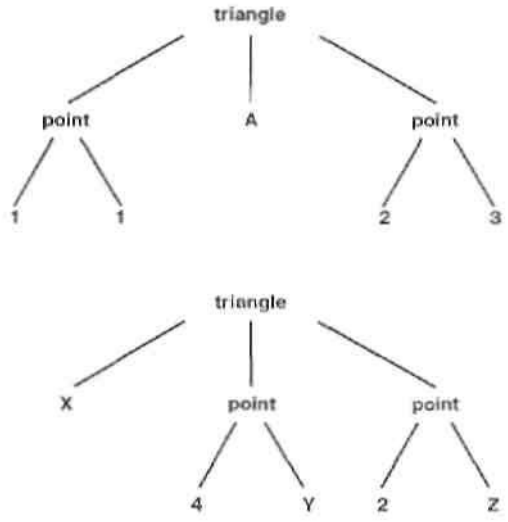

Рис. 2.7, Согласование термов запроса `triangle(point (1,1), A, point (2,3)) = triangle ( X, point(4, Y) , point (2, Z) )`

Весь процесс согласования завершается успешно, поскольку успешными оказываются все операции согласования в этой последовательности. Результирующая конкретизация выглядит следующим образом:

x = point (1,1)

A =point (4, Y)

Z = 3

Следующий пример показывает, что даже одна только операция согласования может использоваться для выполнения некоторых интересных вычислений. Вернемся к примеру простых геометрических объектов (см. рис, 4) и определим фрагмент программы для распознавания горизонтальных и

вертикальных отрезков прямых. *Вертикальность* является свойством отрезков, поэтому такое свойство можно формализовать в языке Prolog как унарное отношение. Пример, приведенный на рис. 8. позволяет понять, как следует формализовать это отношение. Отрезок является вертикальным, если координаты X его конечных точек равны; никакие иные ограничения на отрезок прямой не накладываются. Свойство *горизонтальности* формулируется аналогичным образом, лишь меняются местами координаты X и Y. Требуемое задание можно выполнить с помощью следующей программы, состоящей из двух фактов:

vertical (seg (point(X,Y), point (X, Y1)).

horizontal (seg (point (X,Y), point(X1,Y)).

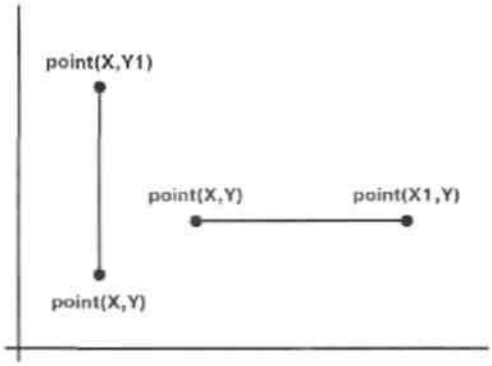

Рис. 2.8. Пример вертикального и горизонтального отрезков прямой

С этой программой может быть проведен следующий диалог:

?- vertical(seg(point(1,1), point(1,2))). - yes

?- vertical(seg(point(1,1), point(2,Y))). no

? - horizontal (seg(point(l, l) , point (2,Y))), Y = 1

На первый вопрос был получен ответ «yes», поскольку цель вопроса согласуется с одним из фактов программы. Для ответа на второй вопрос не удалось найти какого-либо согласования. При ответе на третий вопрос переменной Y было присвоено значение 1 в результате согласования с фактом о горизонтальных отрезках.

Примером более общего вопроса к этой программе может служить вопрос о том, существуют ли какие-либо вертикальные отрезки, которые начинаются в точке (2,3)?

?- vertical (seg(point(2,3),P)).

P =point(2,Y)

Этот ответ означает, что таковыми являются любые отрезки прямой, оканчивающиеся в любой точке (2,Y), иными словами, оканчивающиеся в любом месте на вертикальной прямой x = 2.

Системе приходится формировать новые имена, чтобы обеспечить возможность переименовать пользовательские переменные в программе. Такая

необходимость возникает по двум причинам: во-первых, в связи с тем, что одно и то же имя в разных предложениях обозначает различные переменные, и, во-вторых, из-за того, что при последовательных применениях одного и того же предложения каждый раз используется его "копия" с новым набором переменных.

Еще одним интересным вопросом к этой программе может служить вопрос о том, существуют ли отрезки, которые являются одновременно вертикальными и горизонтальными.

?- vertical(S), horizontal (S). s = seg(point(X,Y), point(X,Y))

В этом положительном ответе системы Prolog на заданный вопрос фактически сказано, что любой отрезок, вырожденный в точку, имеет свойство быть вертикальным и горизонтальным одновременно. И в этом случае ответ был выработан просто в результате согласования. Как и в предыдущем случае, вместо имен переменных X и Y в ответе могут появиться некоторые имена, сформированные внутри программы.

2 Декларативное значение программ Prolog

Как было сказано выше, могут рассматриваться два значения программ Prolog: декларативное и процедурное. Далее будут приведены более формальные определения декларативного и процедурного значений программ в основном подмножестве языка Prolog. Но вначале снова проанализируем различие между этими двумя значениями. Рассмотрим следующее предложение:

P :- Q, R.

Здесь P, Q и R имеют синтаксис термов. Ниже приведены некоторые альтернативные декларативные прочтения этого предложения.

P является истинным, если Q и R истинны.

Из Q и R следует P.

Два альтернативных процедурных прочтения этого предложения приведены ниже.

Чтобы решить проблему P, вначале необходимо решить подпроблему Q, а затем – подпроблему R.

Для достижения цели P вначале необходимо достичь цель Q, а затем - R.

Итак, различие между декларативными и процедурными прочтениями состоит в том, что последнее определяет не только логические соотношения между головой предложения и целями в его теле, но и задает порядок, в котором должны обрабатываться цели.

Рассмотрим формализованное определение декларативного значения.

Декларативное значение программ позволяет установить, является ли заданная цель истинной, а в случае положительного ответа - при каких

значениях переменных она является истинной. Чтобы точно определить декларативное значение, необходимо ввести понятие экземпляра предложения. *Экземпляром предложения* С называется предложение С, в котором вместо каждой из его переменных подставлен некоторый терм. *Вариантом предложения* С называется такой экземпляр предложения С, где вместо каждой переменной подставлена другая переменная. Например, рассмотрим следующее предложение: hasachild(X) :- parent(X, Y).

Ниже приведены два варианта этого предложения.

hasachild(A) :- parent(A, 3).

hasachildl(X1) :- parent(XI, X2) .

Экземплярами этого предложения являются следующие:

hasachild(peter) :- parent(peter, 2).

hasachild(barry) :- parent(barry, small(Caroline)).

Если даны некоторая программа и цель С, то в декларативном значении неявно содержится приведенное ниже утверждение.

Цель G является истинной (т.е. достижимой, или логически следующей иэ программы), если и только если:

1. в программе имеется предложение С, такое, что:

существует экземпляр I предложения С, такой, что

а) голова I идентична цели G и

б) все цели в теле I являются истинными.

Из этого определения можно вывести понятие вопроса Prolog следующим образом. Как правило, вопросом в системе Prolog является список целей, разделенных запятыми. Список целей является истинным, если все цели в списке являются истинными для одной и той же конкретизации переменных. Значения переменных становятся результатом наиболее общей конкретизации.

Поэтому запятая между целями обозначает конъюнкцию целей, при которой все они должны быть истинными. Но Prolog допускает также дизъюнкцию целей, при которой должна быть истинной лишь любая из целей. Дизъюнкция обозначается точкой с запятой. Например, предложение

P :- Q; R

имеет прочтение: P является истинным, если истинно Q или R. Поэтому значение этого предложения аналогично значению двух следующих предложений, вместе взятых:

P :- Q.

P :- R.

Запятая связывает элементы предложения сильнее, чем точка с запятой. Поэтому предложение

P :- Q, R; S, T, U.

может рассматриваться следующим образом:

P :- (Q, R); (S, T, U).

и означает то же, что и предложения

P :- (Q, R).

P :- (S, T, U).

3 Процедурное значение

Процедурное значение определяет, каким образом Prolog отвечает на вопросы. *Получить ответ на вопрос* означает попытаться достичь целей, заданных в списке. Их можно достичь, если переменные, которые встречаются в целях, могут быть конкретизированы таким образом, что цели логически следуют из программы. Поэтому процедурное значение Prolog определяет процедуру выполнения целей в списке применительно к заданной программе. Выражение "выполнить цель" означает попытаться ее достичь.

Обозначим процедуру выполнения целей как execute. Как показано на рис. 1, входы и выходы этой процедуры являются следующими:

• входы - программа и список целей;

• выходы - индикатор успеха/неудач и и конкретизация переменных.

Рис.1

Значение двух выходных результатов описано ниже.

1. Индикатор успеха/неудачи принимает значение "yes", если цели являются достижимыми, и "no" "в противном случае. Принято говорить, что "yes" указывает на успешное завершение, а "no" - на неудачное.

2. Конкретизация переменных вырабатывается только в случае успешного завершения; в случае неудачи конкретизация отсутствует.

Конкретные операции процесса выполнения цели иллюстрируются на примере, приведенном в листинге 1.

Для выполнения следующего списка целей G1, G2, ..., Gm процедура

execute осуществляет описанные ниже действия.

Если список целей пуст, то завершается успешно.

Если список целей не пуст, то продолжает работу со (следующей) операции, называемой "SCANNING".

SCANNING. Сканировать предложения в программе от начала до конца до тех пор, пока не будет обнаружено первое предложение C, такое, что голова C согласуется с первой целью G1. Если такое предложение отсутствует, процедура оканчивается неудачей.

Если есть такое предложение C в форме

H :- B1,..., Bn.

то процедура переименовывает переменные в C для получения варианта C предложения C, такого, что C и список G1,..., Gm не имеют общих переменных. Допустим, что C имеет вид

H' :- B1',..., Bn'.

Процедура согласовывает цель G1 и голову предложения H'; допустим, что результирующей конкретизацией переменных является S.

В списке целей Gl, G2, ..., Gm процедура заменяет цель Gl списком B1',..., Bn', что приводит к получению нового списка целей:

B1',..., Bn', G2, ..., Gm

(Следует отметить, что если предложение C представляет собой факт, то n=0 и новый список целей становится короче, чем первоначальный; такое сокращение списка целей может в конечном итоге привести к получению пустого списка и, тем самым, к успешному завершению.)

Затем процедура подставляет вместо переменных в новом списке целей новые значения, которые заданы в конкретизации S, что приводит к получению еще одного списка целей: B1", . . ., Bn" , G2' . . . .Gm'

После этого происходит переход к выполнению (рекурсивно, с помощью той же процедуры) этого нового списка целей. Если выполнение этого нового списка целей завершается успешно, то выполнение первоначального списка целей также завершается успешно. Если же выполнение нового списка целей оканчивается неудачей, то происходит отказ от этого нового списка целей и возврат через всю программу к операции SCANNING. Сканирование продолжается с предложения, которое непосредственно следует за предложением C (C - это предложение, которое использовалась перед этим), и осуществляется попытка достичь успешного завершения с помощью некоторого другого предложения.

Листинг 2.1. Пример, иллюстрирующий процедурное значение Prolog

```
big( bear).          % Предложение 1
elephant).           % Предложение 2
small( cat).         % Предложение 3
brown(bear).         % Предложение 4
black(cat).          % Предложение 5
gray(elephant) .     % Предложение 6
dark(Z) :- black(Z). % Предложения 7-8 все звери с черной или коричневой
dark(Z) :- brown(Z).% шерстью имеют темную окраску
? - dark(X), big(X).  % Какой из зверей большой и имеет темную окраску?
```

Трассировка выполнения

1) Начальной список целей: dark(X), big(X) .

2) Сканируем программу сверху вниз, отыскивая предложение, голова которого согласуется с первой целью, dark(X). Обнаружено предложение 7:

dark(Z) :- black (Z) .

Подставляем вместо первой цели конкретизированное тело предложения 7, что приводит к получению нового списка целей:

black(X), big(X)

3) Сканируем программу, чтобы найти согласование с целью black(X). Найдено предложение 5: black(cat). Это предложение не имеет тела, поэтому список целей после соответствующей конкретизации сокращается до big(cat)

4) Сканируем программу, чтобы найти цель big(cat) – неудачно. Поэтому возвратщаемся к шагу (3) и отменяем конкретизацию x=cat. Теперь список целей снова принимает вид black(X), big(X).

Продолжаем сканирование программы ниже предложения 5. Не обнаружено ни одного предложения. Поэтому возвращаемся к шагу (2) и продолжаем сканирование ниже предложения 7. Обнаружено предложение 3: dark(Z) :- brown(Z). Подставляем brown(X) вместо первой цели в списке целей, что приводит к получению списка brown(X), big(X) .

5) Сканировать программу, чтобы согласовать цель brown(X), что приводит к обнаружению цели brown(bear). Это предложение не имеет тела, поэтому список целей сокращается до big(bear).

6) Сканируем программу, что приводит к обнаружению предложения big(bear). Оно не имеет тела, поэтому список целей сокращается до пустого. Это указывает на успешное завершение, и соответствующая конкретизация переменной имеет вид

X = bear

Здесь уместно привести несколько дополнительных замечаний, касающихся используемой формы представления процедуры execute. Прежде всего, в ней не было явно описано, как вырабатывается конечная результирующая конкретизация переменных. К успешному завершению привела именно конкретизация 5, которая, возможно, могла быть в дальнейшем уточнена с помощью дополнительных конкретизации, выполненных во вложенных рекурсивных вызовах процедуры execute.

При каждом неудачном завершении рекурсивного вызова execute выполнение возвращается к операции SCANNING и работа продолжается с того предложения С программы, которое использовалось перед этим в последнюю очередь. А поскольку использование предложения С не привело к успешному завершению, система Prolog вынуждена предпринять попытку перейти к обработке альтернативного предложения. При этом фактически происходит следующее: система Prolog отказывается от всей этой части неудачного выполнения и возвращается к той точке (к предложению С), с которой началась данная неудачная ветвь выполнения. После того как процедура возвращается к определенной точке, отменяются все конкретизации переменных, которые были выполнены после этой точки. Такая организация работы гарантирует, что Prolog систематически исследует все возможные альтернативные пути выполнения до тех пор, пока не будет найден тот из них, который в конечном итоге приведет к успеху, или пока не будет продемонстрировано, что все эти путл завершаются неудачей.

Как уже было сказано, даже после некоторого успешного завершения пользователь может вынудить систему вернуться к поиску дополнительных решений. В приведенном выше описании процедуры execute эта деталь была опущена.

Безусловно, в фактических реализациях Prolog необходимо добавить к процедуре execute несколько других усовершенствований. Одно из них состоит в сокращении объема сканирования предложений программы для повышения эффективности. Поэтому в практических реализациях Prolog осуществляется сканирование не всех предложений программы; рассматриваются только предложения, касающиеся данного отношения в текущей цели.