1 Интерпретатор SWI-Prolog

Запуск интрепретатора осуществляется командой `swipl-win`. Если в системе не прописан путь к файлу `swipl-win.exe` (обычно `C:\Program Files (x86)\swipl\bin`), то запуск нужно осуществлять с указание полного пути к этому файлу (как показано ниже).

```
C:\Users\contest>"C:\Program Files (x86)\swipl\bin\swipl-win"
Welcome to SWI-Prolog (Multi-threaded, 32 bits, Version 6.6.6)
Copyright (c) 1990-2013 University of Amsterdam, VU Amsterdam
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

For help, use ?- help(Topic). or ?- apropos(Word).

1 ?-
```

Приглашением интерпретатора для ввода команды является `?-`.

Команда считается завершенной если Enter нажат после ввода точки. После ввода команды система анализирует её, выполняет и выводит результат выполнения на терминал.

Образец диалога:

```
?- 2 > 5.
false.
```

Здесь пользователь вводит фразу `2 > 5.`, что означает «Больше ли 2 чем 5?». Система производит оценку и выводит сообщение о результате: `false.`.

```
?- write('Hello'), nl.
Hello
true.
```

Здесь первая строка вывода на экран является результатом работы введенных предикатов `write` и `nl`, а вторая строка — это результат оценивания запроса Prolog-интерпретатором. Значения `true` указывает на то, что цель, которая была поставлена, успешно решена.

```
?- member(5, [1,3,6,5]).
true.
```

Здесь системе ставится вопрос: является ли число `5` элементом списка `[1,3,6,5]`, на что система отвечает, что является.

Последовательность символов без пробелов, начинающаяся со строчного символа, или последовательность символов, заключенная в апострофы, считается именем факта (команды, функтора). Имя переменной должно начинаться с заглавной буквы или со знака подчеркивания.

Переменные могут участвовать в запросах к системе. Например вопрос

```
?- X = 5.
X = 5.
```

означает: «Каким должен быть X, чтобы выполнялось равенство"?», на что система отвечает, что X должен быть равным 5.

Если система заканчивает свой вывод точкой, это означает, что действие завершено и все что нужно было сделать — сделано. Если система находит несколько ответов на вопрос, то после вывода очередного ответа на экран она не выводит точку, а ожидает действия пользователя: при нажатии «;» осуществляется поиск следующего ответа; при нажатии Enter заканчивается поиск ответов на вопрос и ставится точка. Например:

```
?- member(X, [1,2,3,4,5]).
X = 1 ;
X = 2 ;
X = 3 ;
X = 4 ;
X = 5.
```

Здесь системе ставится вопрос: «Чему должен равняться X, чтобы X был элементом списка?». Система по очереди дает ответы на вопрос, делая между ответами паузы для дальнейшего выбора действия пользователя.

При оценивании факта, имеющего параметры, параметры перечисляются в круглых скобках, через запятую. При этом не допускается пробелов между именем факта и открывающейся круглой скобкой. В противном случае система выдаст сообщение об ошибке.

```
?- write (123).
ERROR: Syntax error: Operator expected
ERROR: write
ERROR: ** here **
ERROR:  (123) .
```

Для включения в тело программы на языке Prolog комментариев следует использовать специальные скобки `/*  */`. Текст, расположенный между ними считается комментарием.

Однострочные комментарии начинаются с символа `%` и продолжаются до конца строки.

В среде SWI-Prolog справку о любом встроенном слове языка можно вызвать прямо из консоли, используя команду `help`, например:

```
?- help.
?- help(nl).
?- help(write).
```

2 Базовые сведения о Prolog

Программа на языке Prolog описывает знания о некоторой предметной области. Эти знания включают в себя факты, описывающие свойства объектов и отношения между ними, и правила, на основании которых могут быть получены новые знания о свойствах объектов и отношениях между ними. Факт состоит из имени отношения и списка объектов, между которыми это отношение существует. Каждый факт должен заканчиваться точкой. Например, факт «Джону нравится Мэри» записывается так:

```
'нравится'('Джон', 'Мэри').
```

Выполнение программы инициализируется запросом. Интерпретатор пытается логически вывести запрос, используя знания о предметной области, содержащиеся в программе, которая является своего рода базой данных (БД) об объектах в задаче. Если существует факт, сопоставимый с целью, указанной в запросе, то интерпретатор отвечает утвердительно. В противном случае будет получен отрицательный ответ. Например, пусть БД содержит следующие факты:

```
'нравится'('Джон', 'Мэри').
'нравится'('Джон', 'пиво').
'нравится'('Мэри', 'кино').
'нравится'('Джон', 'кино').
```

Запрос пишется после подсказки ?- и должен заканчиваться точкой. Например, на эти запросы будут получены следующие ответы:

```
?-''нравится'('Джон', 'деньги').
false
?-'нравится'('Мэри', 'Джон').
false
?-'нравится'('Мэри', 'кино').
true
?-'президент'('Обама', 'США').
false
```

На второй запрос был получен отрицательный ответ, так как из того, что Джону нравится Мэри, не следует, что Мэри нравится Джон. Остальные отрицательные ответы связаны с ограниченностью БД, в которой не отражаются связи, возможно существующие в действительности. Чтобы узнать все объекты, которые нравятся Джону, нужно указать в запросе вместо имени конкретного объекта переменную.

```
?- 'нравится'('Джон', X).
X = 'Мэри';
X = 'пиво';
X = 'кино';
false
```

Интерпретатор сопоставляет цель в запросе с фактами в БД, при этом переменная получает значение аргумента отношения, стоящего в соответствующей позиции.

Используя конъюнкцию целей, можно строить более сложные запросы. При этом цели в запросе перечисляются через запятую, а для получения утвердительного ответа необходимо согласование всех целей в запросе.

Например, запрос «Существует ли что-либо, что нравится и Джону, и Мэри?» выглядит так:

```
?- 'нравится'('Джон', X), 'нравится'('Мэри', X).
```

Если факт — это отношение, которое безусловно истинно, то правило — это отношение, которое является истинным, только если выполняется некоторое условие. Правило состоит из головы, которая определяет новое свойство или отношение объектов, и тела, которое содержит условие. При попытке сопоставления цели в запросе и головы правила выполняется согласование целей в теле правила. Если это согласование успешно, то отношение, задаваемое головой правила, истинно.

Правило должно заканчиваться точкой, а голова от тела правила отделяется обозначением «:-», которое читается как «если». Например, правило «Мэри нравится любой, кому нравится кино» записывается как:

```
'нравится'('Мэри', X) :- 'нравится'(X, 'кино').
```

Пусть БД содержит факты:

```
'мужчина'('Адам').
'мужчина'('Каин').
'мужчина'('Авель').
'женщина'('Ева').
'родитель'('Адам', 'Каин').
'родитель'('Адам', 'Авель').
'родитель'('Ева', 'Каин').
'родитель'('Ева', 'Авель').
```

Правило «X приходится сестрой Y, если X является женщиной, и X и Y имеют общих родителей» на языке Prolog выглядит следующим образом:

```
сестра(X, Y) :- женщина(X), родитель(Z, X), родитель(Z, Y).
```

Данное определение может привести к ошибке, так как Prolog не требует различия объектов, обозначаемых разными именами, которое неявно подразумевается в определении на естественном языке. Поэтому эта проверка должна быть сделана явно:

```
сестра(X, Y) :- женщина(X), родитель(Z, X), родитель(Z, Y), X \= Y.
```

3 Списки

Списки в языке Пролог представляются в виде последовательности элементов, разделенных запятыми, в квадратных скобках. Например, список может быть записан как `[a, b, c]`.

Для разделения списка на голову и хвост в Prolog используется специальная форма для представления списка с головой `X` и хвостом `Y`, которая записывается как `[X | Y]`. Перед вертикальной чертой можно перечислить любое число первых элементов списка, после вертикальной черты указывается одно значение, соответствующее хвосту списка. Например, список `[a, b, c]` может быть записан следующими способами:

```
[a | [b, c]]
[a, b | [c]]
[a, b, c | []]
[a | [b | [c | []]]]
```

4 Сопоставление

Сопоставление (унификация) является наиболее важной операцией в языке Prolog. Сопоставление выполняет сравнение двух термов на равенство, при этом неконкретизированные переменные получают значения, при которых термы становятся идентичными.

Рассмотрим запрос:

```
?- дата(D, январь, Y) = дата(1, M, Z).
D = 1,
Y = Z,
M = январь.
```

Если переменная используется в предложении дважды, причем один раз как аргумент предиката сопоставления, то от явного сопоставления можно избавиться, заменив переменную вторым аргументом предиката сопоставления. Например, следующие два предложения эквивалентны:

```
список(X) :- X = [_|_].
список([_|_]).
```

В Прологе существует также противоположный предикат `X \= Y`, который истинен только в случае, если терм `X` не сопоставим с термом `Y`. При использовании этого предиката в программе рекомендуется, чтобы все переменные в термах `X` и `Y` на момент согласования цели были конкретизированными, иначе результат будет зависеть от порядка целей в программе:

```
?- X = a, Y = b, X \= Y.
X = a,
Y = b.
?- X = a, X \= Y, Y = b.
false.
```

Иногда требуется проверить точное равенство двух термов, включая соответствие расположения и идентичность неконкретизированных переменных. Это осуществляется с помощью встроенного предиката равенства (идентичности) `X == Y`. Этот предикат не выполняет конкретизации переменных, неконкретизированная переменная не равна никакому объекту, кроме другой неконкретизированной переменной, уже сцепленной с ней. Предикат равенства остается истинным, какое бы значение не получила в ходе дальнейшего вывода неконкретизированная переменная, входящая в терм.

```
?- f(a, X) == f(a, X).
true.
?- X == a.
false.
?- X = a, X == a.
X = a.
?- X == Y.
false.
?- X = Y, X == Y.
X = Y.
```

Противоположный предикат `X \== Y` истинен только в случае, если терм `X` не равен терму `Y`.

5 Арифметические выражения

Роль арифметических предикатов, встроенных в Пролог, заключается в обеспечении интерфейса с арифметическими возможностями компьютера. Так как аргументы предикатов в Прологе не вычисляются, а передаются в виде термов, в следующем запросе структура `2 + 2` не сопоставима с числом `4`:

```
?- 2 + 2 = 4.
false.
```

Поэтому необходимые арифметические вычисления должны быть произведены до момента согласования цели. Для этого используется предикат `X is Y`, который интерпретирует терм `Y` как арифметическое выражение, вычисляет его и сопоставляет результат с термом `X`. Выражение должно состоять из чисел, стандартных операций сложения (`+`), вычитания (`-`), умножения (`*`), деления (`/`), остатка от деления (`mod`), деления нацело (`//`), возведения в степень (`^`) и математических функций (`sqrt`, `integer`, `random`, `sin`, `cos` и т. д.). Если терм `Y` не является корректным арифметическим выражением, содержит неконкретизированные переменные, атомы, неизвестные операции или функции, то выводится сообщение об ошибке.

```
?- X is 2 * 2.
X = 4.
?- 4 is 2 * 2.
true.
```

Другими арифметическими предикатами являются предикаты сравнения `X =:= Y`, `X =\= Y`, `X > Y`, `X < Y`, `X =< Y`, `X >= Y`, которые вычисляют термы `X` и `Y` как арифметические выражения и сравнивают численные результаты.

```
?- 5 * 3 =:= 90 / 6.
true.
?- 7.5 < 3 * 3.
true.
```

Пролог позволяет формировать сложные логические выражения. Простейшими логическими предикатами являются `true` (истина) и `fail` (ложь, неудача). Согласование цели `true` всегда успешно. Согласование цели `fail` всегда неудачно. Для конъюнкции целей используется предикат «`,`» `(X, Y)`, а для дизъюнкции — предикат «`;`» `(X; Y)`. Приоритет у оператора «`,`» выше, чем у оператора «`;`», поэтому лишние скобки можно опускать. Например, выражение `A, B; C, D` интерпретируется как `(A, B); (C, D)`. Для отрицания используется предикат `not(X)`. Так как запятая служит как для конъюнкции целей, так и для разделения аргументов, требуются дополнительные скобки, если аргумент `not` не является элементарным выражением. Например, нужно писать `not((A, B))`, а не `not(A, B)`.

6 Отсечение

Процесс доказательства в Прологе основан на переборе в глубину. Сначала выбирается первая по порядку альтернатива, которая может быть использована для доказательства цели. Если происходит возврат в точку выбора (в случае неуспеха доказательства), то выбирается следующая альтернатива, и так далее. Процесс перебора вариантов будет продолжаться до тех пор, пока не будет доказан запрос. Кроме того, пользователь может инициировать процесс возврата после получения решения, если он захочет получить другой вариант решения. Предикат отсечения, обозначаемый символом «`!`», позволяет уменьшить пространство поиска и сократить перебор, отбрасывая неиспользованные альтернативы. Выполнение отсечения приводит к следующим результатам.

Отбрасываются все предложения для согласуемой цели, расположенные после правила, содержащего отсечение. Отбрасываются все альтернативные решения конъюнкции целей, расположенных в предложении левее отсечения. С другой стороны, отсечение не влияет на цели, расположенные правее его. В случае возврата они могут порождать альтернативные решения. Если доказательство этих целей не будет успешным, происходит возврат к последнему выбору, сделанному перед выбором предложения, содержащего отсечение.

Пусть программа содержит следующее предложение:

```
a :- b, c, !, d, e.
```

После выполнения предиката `!` (в случае успешного доказательства целей `b` и `c`), в случае неуспеха доказательства целей `d` и `e` альтернативы для `c`, `b`, `a` не рассматриваются, и доказательство цели `a` завершается неуспехом.

Предикат отсечения можно сравнить с барьером, через который не может перейти процесс возврата.

Можно выделить три основных случая использования отсечения.

6.1 Подтверждение правильности выбора правила

Обычно каждое предложение определения предиката соответствует аргументам определенного вида. Отсечение фиксирует выбор правила, повышает эффективность программы за счет отбрасывания неподходящих альтернатив. Например, предикат для нахождения максимума можно определить так:

```
максимум(X, Y, X) :- X >= Y, !.
максимум(X, Y, Y) :- X < Y, !.
```

Цель максимум `(5, 2, X)` имеет только одно решение, и отсечение позволяет не проверять второе правило в случае возврата. Однако отсечение делает программу более детерминированной, менее гибкой, чем программа без отсечения. Например, если предикат сцепления двух списков определить так:

```
сцепить([], L, L) :- !.
сцепить([H | L1], L2, [H | L3]) :- сцепить(L1, L2, L3).
```

то его нельзя будет применить для получения всех вариантов расщепления списка:

```
?- сцепить(X,Y,[a,b,c]).
X = [],
Y = [a, b, c].
```

Но это не существенно, если с самого начала предполагается использовать программу лишь одним способом. Программист, зная алгоритм вычисления истинности целей, может использовать

отсечение и для устранения условий, которые следуют из безуспешности применения предыдущих правил. Например, в программе

```
максимум(X, Y, X) :- X >= Y, !.
максимум(X, Y, Y).
```

условие `X < Y`, опущенное во втором предложении, следует из неудачи при сравнении `X >= Y`. Устранение дополнительных проверок с очевидным результатом повышает эффективность программы, но поведение такой модифицированной программы не всегда корректно. Например, следующий запрос дает неправильный ответ:

```
?- максимум(5, 2, 2).
true.
```

Чтобы избавиться от ошибки, требуется неявное сопоставление первого и третьего аргумента предиката сделать явным и выполнить его после отсечения.

```
максимум(X, Y, Z) :- X >= Y, !, Z = X.
максимум(X, Y, Y).
```

В результате программа стала менее понятной. Кроме того, правильность программы стала зависеть от порядка предложений в программе, что противоречит смыслу логического программирования.

6.2 Прекращение доказательства цели

Предикат может быть заведомо ложным для некоторых аргументов или может быть неприменим к аргументам определенного вида. Удобно выделять такие исключения в отдельные правила, уменьшая таким образом сложность определения. Это можно сделать с помощью комбинации «отсечение-неудача». Например, для пустого списка невозможно вычислить среднее значение:

```
среднее([], R)            :- !, fail.
среднее(L, R)             :- среднее(L, 0, 0, R).
среднее([], S, K, R)      :- R is S / K.
среднее([H | T], S, K, R) :- S1 is S + H, K1 is K + 1, среднее(T, S1, K1, R).
```

Встроенный предикат `not(G)`, который истинен, если цель `G` не удалось согласовать с БД, можно определить, используя комбинацию «отсечение-неудача»:

```
not(G) :- G, !, fail.
not(G).
```

Реализация в Прологе отрицания как неудачи при доказательстве цели, может приводить к отрицательным результатам, даже если решение существует. Рассмотрим следующую программу.

```
студент(билл).
студент(джон).
женат(джон).
неженатый_студент(X) :- not(женат(X)), студент(X).
?- неженатый_студент(X).
false.
```

Ответ на запрос отрицательный, несмотря на то, что решение `X = билл` логически следует из правила и фактов. Это связано с тем, что цель `not(женат(X))` ложна из-за решения `X = джон`. Проблема может быть устранена перестановкой целей в теле правила. В большинстве случаев рекомендуется, чтобы цель под отрицанием не содержала неконкретизированных переменных. Такое же правило относится и к предикату `\=`.

6.3 Завершение последовательности порождения и проверки вариантов

Часто цели в Прологе могут быть согласованы множеством различных способов и порождают много возможных решений при возврате. С помощью отсечения можно завершить порождение альтернативных решений, если в них нет необходимости, или существует только единственное решение. Отношение «муж» для БД о родственных связях можно определить так:

```
'муж'(X, Y) :- 'мужчина'(X), 'женщина'(Y), 'родитель'(X, Z), 'родитель'(Y, Z).
```

Но при наличии нескольких детей появляются ненужные множественные решения:

```
?- 'муж'('Адам',X).
X = 'Ева' ;
X = 'Ева' ;
X = 'Ева' ;
false.
```

от которых можно избавиться с помощью отсечения:

```
'муж'(X, Y) :- 'мужчина'(X), 'женщина'(Y), 'родитель'(X, Z), 'родитель'(Y, Z), !.
```

```
?- 'муж'('Адам',X).
X = 'Ева'.
```

Надежное использование отсечения возможно только в случае, если имеется четкое представление о том, как программа будет использоваться. При появлении новых способов использования программы необходимо пересмотреть все случаи употребления отсечения.