Парадигма логического программирования. Язык Пролог

Язык Пролог

- 1972, Ален Колмероэ
- Язык Пролог – в широко распространенная реализация принципов логического представления знаний.
- Реализует логику предикатов первого порядка.
- Декларативный язык (отвечает на вопрос «что это?», а не «как это сделать?» как в процедурных языках)

Критика Пролога

- В Прологе реализуется лишь логическое представление знаний и лишь достоверный вывод, в то время как в большинстве случаев требуется использовать рассуждения в условиях неопределенности.
- Недостаточное удобство и гибкость, из-за которых создание программных продуктов с использованием этого языка часто оказывается более трудоемким и требующим специфической квалификации разработчиков, чем с использованием языков высокого уровня общего назначения.

Особенности Пролога

- Для решения ряда задач Пролог оказывается гораздо более эффективным, чем, например, C++ или Java.
  - внутренне присущий Прологу декларативный стиль представления знаний облегчает более легкую поддержку программ в течение их жизненного цикла;
- В связи с этим Пролог следует рассматривать в качестве специализированного языка программирования, который может использоваться в экспертных системах или для быстрого создания прототипов интеллектуальных систем, и не противопоставлять языкам общего назначения, а использовать как дополнительный инструмент.

Синтаксис

- Преимущественно состоит из списка логических утверждений, которые можно разделить на факты и общие правила.
- Эти утверждения составляются из имен предикатов, переменных и их значений, которые представляют собой символьную строку, начинающуюся с буквы, а также совокупности логических операций.
- В конце каждого утверждения ставится точка ".".

Синтаксис

- Переменные обязательно начинаются с прописной буквы, поэтому все значения должны начинаться со строчной буквы.
- Кванторы существования и всеобщности в Прологе опускаются: хотя они фактически используются, в явном виде в утверждениях (общих правилах, в которых фигурируют переменные) они не используются.
- Для предикатов, переменных и их значений отсутствуют объявления, как это имеет место в обычных языках.

Логические операции в Прологе

Имя операции	Логическая операция	Обозначение в Прологе
и	$\wedge$	,
или	$\vee$	;
если	$\Leftarrow$	:–
не	$\neg$	not

Пример утверждений

- Список фактов, не содержащих переменных:

```
man(serg).
man(alex).
father(serg, alex).
mother(kat, serg).
```

- Эти факты говорят, что предикаты "man", "father", "mother" истинны при указанных значениях аргументов.
  - Предположение о замкнутости базы знаний (все предикаты будут ложны для всех значений переменных, для которых не было указано обратное).

Общие правила

- Общие правила в Прологе записываются так же, как и факты, но в них присутствуют переменные. При этом подразумевается, что общее правило верно при всех значениях всех входящих в него переменных.
- p1(X) :– p2(X,Y) означает логическое выражение $(\forall X, Y) p_2(X, Y) \Rightarrow p_1(X)$

Общие правила

- Поскольку в Прологе подобные выражения используются для получения значения p1(X) при некотором X, то это выражение удобнее заменить выражением $(\forall X)p_1(X) \Leftarrow (\exists Y)p_2(X,Y)$
- p1(X) истинно, если существует хотя бы одно значение Y, при котором p2(X,Y) истинно.
- Такая форма гораздо удобнее для вывода, поскольку выражение p2(X,Y)=>p1 (X) истинно при любых значениях p1 (X), если при данном Y предикат p2(X,Y) ложен.
- То есть на основе истинности всего выражения мы не можем установить истинность p1 (X).

Общие правила

- Если дано правило

  p1(X,Y) :– p2(X)

то оно означает вовсе не выражение

$$(\forall X) p_2(X) \Rightarrow (\exists Y) p_1(X,Y)$$

а выражение

$$(\forall X) p_2(X) \Rightarrow (\forall Y) p_1(X,Y)$$

поскольку операция «<=» несимметрична.

Примеры

- father(X,Y) :– man(X), parent(X,Y).
  - $(\forall X, Y) man(X) \wedge parent(X,Y) \Rightarrow father(X,Y)$
  - Если некто *X* является мужчиной и родителем *Y*, то *X* – отец *Y*.
- parent(X,Y) :– mother(X,Y); father(X, Y).
  - $(\forall X, Y) mother(X,Y) \vee father(X,Y) \Rightarrow parent(X,Y)$
  - *X* является родителем *Y*, если *X* – отец или мать *Y*.

Примеры

- grandfather(X,Y) :– father(X, Z), parent(Z,Y).
  - $(\forall X, Y)(grandfather(X,Y) \Leftarrow (\exists Z)\, father(X,Z) \wedge parent(Z,Y))$
  - *X* является дедушкой *Y* только тогда, когда *X* является отцом (некоторого) родителя *Y*.
- woman(X) :– mother(X,Y).
  - $(\forall X)(woman(X) \Leftarrow (\exists Y) mother(X,Y))$
  - Если *X* приходится кому-то матерью, то она является женщиной.
- brother(X,Y) :– man(X), parent(Z, X), parent(Z, Y).
  - $(\forall X, Y)(brother(X,Y) \Leftarrow (\exists Z) man(X) \wedge parent(Z,X) \wedge parent(Z,Y))$
  - Если у *X* и *Y* общий родитель и *X* – мужчина, то *X* – брат *Y*.

Плохой пример

- mother(X,Y) :– woman(X).
  - $(\forall X)(woman(X) \Rightarrow (\forall Y) mother(X,Y))$
  - Если *X* является женщиной, то она мать любого *Y*.

Программа на языке Пролог

- Представляет собой список общих правил и фактов и не наделена функциональностью, а декларативно представляет некие знания.
- За исполнение программы отвечает интерпретатор, который взаимодействует с пользователем (в качестве которого также вполне может выступать и программа не каком-то другом языке), интерактивно отвечая на его запросы.
- Запрос к интерпретатору также представляется в виде логического выражения, истинность которого требуется установить.

Пример программы

```
mother(X, Y) :– woman(X), parent(X, Y).
father(X,Y) :– man(X), parent(X,Y).
grandfather(X,Y) :– father(X, Z), parent(Z,Y).
man(serg).
man(alex).
woman(kat).
parent(serg, victor).
parent(kat, victor).
parent(victor, alex).
```

Пример программы

- Посмотрим на следующие запросы, начинающиеся с приглашения "?–", на которые интерпретатор возвращает некоторый ответ:
  - ?– father(serg, victor).
    - Yes
  - ?– grandfather(serg, alex).
    - Yes
  - ?– mother(kat, alex).
    - No
  - ?– father(victor, alex).
    - No

Выводы

- Для обработки этих запросов интерпретатору необходимо выполнить логический вывод, а не просто обратиться к базе правил.
- Видно, как действует предположение о замкнутости базы знаний: поскольку в базе нет фактов об истинности parent(kat, alex) и man(victor), то последние два запроса возвращают "Нет".

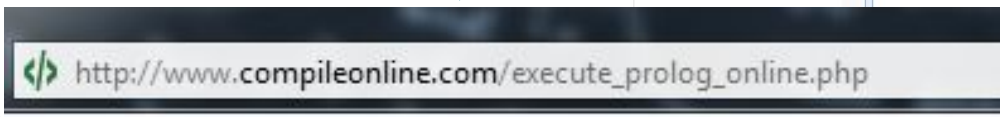
http://www.compileonline.com/execute_prolog_online.php

compileonline.com - Execute Prolog Script Online (GNU Prolog 1.4.0)

Home | Web Editors | About | Help

Execute Script | main.pr | input.txt | Default Ace Editor

```
1 mother(X, Y) :- woman(X), parent(X, Y).
2 father(X, Y) :- man(X), parent(X, Y).
3 grandfather(X, Y) :- father(X, Z), parent(Z, Y).
4 man(serg).
5 man(alex).
6 woman(kat).
7 parent(serg, victor).
8 parent(kat, victor).
9 parent(victor, alex).
```



Result | Download Files

Executing the program....

$gprolog --consult-file main.pr

```
GNU Prolog 1.4.0
By Daniel Diaz
Copyright (C) 1999-2011 Daniel Diaz
compiling /web/com/141520231617660/main.pr for byte code...
/web/com/141520231617660/main.pr compiled, 10 lines read - 1662 bytes written, 12 ms

X = victor

yes
```

Вывод результата

Место для программы (аксиом)

Место для ввода переменных (или других аргументов командной строки)

Место для запроса (выражения, истинность которого нужно проверить)

Command Line Arguments: X = victor

STDIN Input: mother(kat, X).

Запросы в виде выражений с переменными

- ?– mother(kat, X).
  - X = victor
  - Yes
- ?– grandfather(X, alex).
  - X = serg
  - Yes
- ?– grandfather(X,Y).
  - X = serg,Y = alex
  - Yes
- ?– grandfather(X, kat).
  - No
- ?– man(X).
  - X = serg
  - Yes

Запросы в виде выражений с переменными

- В подобного рода запросах предполагается квантор существования для входящих в него переменных.
- Выражение истинно, если найдется хотя бы одно подходящее значение X (как именно обозначать переменные в запросе, не имеет значения).
- В действительности, при нахождении в базе первого значения, для которого введенное выражение истинно, интерпретатор спрашивает пользователя, продолжать ли поиск. Поиск продолжается при нажатии команды ";" (или) до тех пор, пока не будет дан ответ "Нет" в связи с исчерпанием всех возможностей.

```
?– man(X).
  X = serg ;
  X = alex ;
  No
```

Запросы в виде выражений с переменными

- Более сложные выражения в запросах также допустимы, например,
  - ?– mother(kat, X), father(Y, X).
    - X = victor, Y = serg
    - Yes
- Этот запрос позволяет определить, кто является отцом сына kat.

Команды для изменения базы

- Интерпретатор не только отвечает на запросы, но также позволяет интерактивно изменять базу правил и фактов, для чего могут использоваться команда **assert** для добавления записи (**asserta** и **assertz** позволяют добавлять записи в начало и конец списка соответственно) и команда **retract** для удаления записи.
- Существует также набор команд интерпретатора для отладки базы и процесса вывода, например, команды **listing**, **trace** и **spy**.

Арифметические выражения

- Сложение "+"
- Вычитание "-"
- Умножение "*"
- Деление "/"
- Остаток от целочисленного деления "mod"

Арифметические выражения

- Для того чтобы арифметическое выражение вычислялось, необходимо использовать оператор is.
- Пример:
  - ?– X is 8+3*12.
    - X = 44
    - Yes
- То есть интерпретатор производит «поиск» такого значения переменной X, при котором введенное выражение истинно.

Арифметические выражения

- Арифметические выражения могут использоваться также и для описания связи переменных, входящих в предикаты.
- Пусть в базе записано следующее правило:
  - f(X, Y, Z) :– Z is X*Y.
- Тогда возможны следующие запросы:
  - ?– f(3, 2, 6).
    - Yes
  - ?– f(5, 7, Z).
    - Z = 35
    - Yes

Арифметические выражения

- Однако запрос вида f(5, Y, 35) является некорректным, поскольку интерпретатор требует, чтобы переменные, входящие в арифметическое выражение после оператора is были определены на момент вычисления.
- Таким образом, поиск в действительности не осуществляется, а происходит простое вычисление.

Пример вычисления факториала

- Поместим в базу следующие правила:
  - fr(1,M) :- M is 1.
  - fr(N,M) :- N> 1, N 1 is N-1, fr(N 1, M 1),M is N*M 1.
- Мы заставляем интерпретатор вычислять значение M на основе такого значение M1, для которого будет истинным предикат fr(N-1, M1).

Пример вычисления факториала

- Несложно убедиться, что этот предикат позволяет вычислять значение факториала:
  - ?– fr(5, X).
    - X = 120
    - Yes
  - ?– fr(8, X).
    - X = 40320
    - Yes
  - ?– fr(3, 8).
    - No

Задание

- Напишите список правил, позволяющих вычислять числа Фибоначчи.

Числа Фибоначчи (решение)

- fibonacci(0,M) :- M is 1.
- fibonacci(1,M) :- M is 1.
- fibonacci(N,M) :- N > 1,
  N1 is N-1, fibonacci(N1,M1),
  N2 is N-2, fibonacci(N2,M2),
  M is M1 + M2.

Списки

- Помимо арифметических операций в Прологе также могут использоваться списки – упорядоченные множества элементов, которые являются чрезвычайно важными.
- Списки в Прологе имеют определенное сходство со списками в Лиспе – другом языке искусственного интеллекта, – основой которого они являются.

Списки

- Список в Прологе задается перечислением своих элементов через запятую внутри квадратных скобок:
  - [] – пустой список;
  - [serg, kat, victor, alex] – четырехэлементный список;
  - [[serg, kat], [victor, alex]] – список из двух элементов, которые сами по себе являются списками.

Списки

- Поскольку в Прологе все основано на предикатах, то для объявления некоторого списка нужно ввести соответствующий предикат.
- Список как элемент базы фактов может выглядеть, например, следующим образом:
  - family([serg, kat, victor, alex]).
  - family([nick, ann, igor]).

Списки: расщепление

- Основной операцией по работе со списками является их расщепление на голову (первый элемент списка) и хвост (оставшиеся элементы списка).
- Эта операция осуществляется заданием шаблонов вида [X|Y], где X – голова списка, а Y – его хвост.

Расщепление списков: примеры

- ?– family([serg|X]).
  - X = [kat, victor, alex]
  - Yes
- ?– family([nick, ann|X]).
  - X = [igor]
  - Yes
- ?– family([X,Y|[victor, alex]]).
  - X = serg, Y = kat
  - Yes

Шаблоны списков

- [X|Y] – список, состоящий не менее чем из одного элемента
- [X, Y] – список, состоящий ровно из двух элементов
- [X, Y|Z] – список, состоящий не менее чем из двух элементов
- [a|X] – список, первым элементом которого является «a».
- [X|[a]] – список, состоящий из двух элементом, второй элемент – «a».

Шаблоны списков

- На основе шаблонов можно создавать общие правила для списков.
- Предикат, проверяющий принадлежность элемента списку:
  - member(X, [X|L]).
  - member(X, [Y|L]) :– member(X, L).
- Если X – голова списка, имеющего хвост L, то X является членом списка XL; если X является членом хвоста списка L, то он является также и членом полного списка YL, где Y – произвольный хвост.

Списки: примеры запросов

- Пусть, к примеру, мы делаем запрос:
  - ?– member(serg, [serg, kat, victor, alex]).
- Тогда интерпретатор, обращаясь к правилу member(X, [X|L]) делает подстановку: X=serg, [X|L]=[serg|kat, victor, alex], подстановка оказывается успешной, и он возвращается истинное значение.

Списки: примеры запросов

- Сделаем запрос
  - ?– member(serg, [kat, serg, victor, alex]).
- Интерпретатор не сможет выполнить унификацию для первого правила и обратится ко второму: member(X, [Y|L]) :– member(X, L). Теперь X=serg, Y=kat, L=[serg, victor, alex]. После этого он будет проверять истинность для member(serg, [serg, victor, alex]).

Списки: примеры запросов

- Мы можем воспользоваться введенными ранее фактами для предиката family и общими правилами для предиката member и организовать такой запрос:
  - ?– member(kat, X), family(X).
    - X=[serg, kat, victor, alex]
    - Yes
- То есть мы сможем получать список по входящему в него элементу.

Списки: встроенные предикаты

- length(L, N) является истинным, если N – длина (количество элементов) списка L.
- Примеры:
  - length(L, X), family(L).
    - L = [serg, kat, victor, alex], X = 4 ;
    - L = [nick, ann, igor], X = 3
    - Yes
  - ?– length(X, 4), family(X).
    - X = [serg, kat, victor, alex]
    - Yes

Списки: встроенные предикаты

- member(X, L) в действительности является встроенным предикатом;
- append(L1, L2, L3) истинен, когда L3 является объединением двух списков L1 и L2;
- last(X, L) истинен, когда X – последний элемент в списке L;
- reverse(L1, L2) истинен, когда L2 – это обращенный список L1.

Прочие возможности Пролога

- Мы рассмотрели лишь самые базовые возможности языка Пролог, которые, в действительности, существенно шире.
- Например, на основе списков и механизмов вывода в Прологе строятся прочие абстрактные типы данных: стеки, очереди, множества.
- Помимо различных структур данных Пролог также поддерживает некоторые механизмы управления поиском, которых мы не касались.
- Современные версии Пролога имеют также обширные библиотеки, например, для выполнения и обработки **http**-запросов, что существенно расширяет сферу применения данного языка.