Learn Prolog Now 翻译 - 第九章 - 语句深究 - 第四节， 操作符

内容提要

- 操作符的属性
- 自定义操作符

正如我们之前看到的，在一些特定的情况下（比如，当进行数字计算时），Prolog允许我们书写比内部表示更加友好的操作符语法。事实上，如我们将要学习到的，Prolog甚至允许我们自定义操作符。这本章中，我们将会学习操作符的属性，及其如何自定义操作符。

操作符的属性

首先从数字运算的例子开始。Prolog会在内部使用这样的表达式：is(11, +(2, 、*(3,3)))，但是我们可以自由地在编程中将函子*和+写在参数之间，从而构成更加友好的表达式：11 is 2 + 3*3。函子能够写在参数之间被称为中缀操作符。Prolog中其他一些中缀操作符的例子是：:-, -->, =, =.., ==等等。除了中缀操作符之外，还有前缀操作符（写在参数之前）和后缀操作符（写在参数之后）。比如，?-是一个前缀操作符，还有就是单独的负号-，代表负数（比如，1 is 3+ -2）。后缀表达式的一个例子就是在C语句中的++，可以用于变量的自增。

当我们学习Prolog的数字运算时，我们已经了解到Prolog可以消除数字运算表达式的歧义。所以当我们写 2 + 3*3的时候，Prolog知道其含义是：2 + (3*3)，而不是 (2 + 3) * 3。但是Prolog是如何知道的？因为每一个操作符都有一定的优先级。+的优先级比*的优先级别更高，所以+能够成为表达式 2 + 3*3的主函子。（这里注意理解Prolog优先级高的含义和我们平时的理解不一致，优先级高在Prolog中是指越外层的函子，比如内部表达式为：+(2, *(3, 3))）。类似地，is的优先级比+的优先级高，所以 11 is 2 + 3*3 会被转换为内部的表达式：is(11, +(2, *(3, 3)))。在Prolog中，优先级使用从0到1200的数字表示；最大的数字，代表最高的优先级。给出一些例子，=的优先级是700，+的优先级是500，*的优先级是400。

如果在一个表达式中存在多个相同优先级的操作符时会发生什么？比如之前我们说查询，2 =:= 3 == =:=(2,3)会使得Prolog报错。Prolog不知道如何解析表达式，是 =:=(2, ==(3, =:=(2,3)))，还是 ==(=:=(2,3), =:=(2,3))？原因是因为== 和 =:=有相同的优先级，而Prolog不能决定正确的括号方式。在这种情况下，显式的括号是必须要程序中提供的。

下面的查询会如何进行？

```
?- X is 2 + 3 + 4.
```

Prolog会困惑吗？完全不会：它工作的很愉快并且得出的正确的答案 X = 9。但是内部采用了哪种括号的方式，是 is(X, +(2, +(3,4)))，还是 is(X, +(+

--Benjamin Yan

(2,3), 4))? 如下面的查询所示，Prolog选择的是第二年方式：

```
?- 2 + 3 + 4 = +(2, +(3, 4)).
false

?- 2 + 3 + 4 = +(+(2, 3), 4).
true
```

这里Prolog会使用+的结合性来消除歧义：+是左结合性的，意味着+右边的表达式的优先级并且小于+本身的优先级，同时左边的表达式的优先级必须等于+本身的优先级。一个表达式的优先级简单地认为和其主操作符的优先级一致，或者当被括号括起来时为0. 3 + 4这个表达式的主操作符是+，所以将 2 + 3 + 4 转换为 +(2, +(3, 4))意味着第一个+右边的表达式的优先级和+本身一致，这是不正确的。它必须要更低才行。

操作符==，=:=被定义为没有结合性，这意味着操作符两边的参数必须要有更低的优先级。所以 2 =:= 3 == =:=(2, 3)是一个错误的表达式，因为无论如何加括号，都会有歧义：2 =:= 3有和=相同的优先级，同时 3 == =:=(2,3)和=:=有相同的优先级。

操作符的类型（中缀，前缀和后缀），它们的优先级，和它们的结合性是Prolog中关于操作符必须了解的知识，这样我们才能够写出符合用户使用习惯，同时满足Prolog内部表达方式的代码。

自定义操作符

除了为特定的一些函子提供了友好的操作符语法外，Prolog也允许自定义操作符。比如，你能够定义一个后缀操作符，is_dead，Prolog允许你在知识库中写出 zed is_dead 来替代标准的 is_dead(zed)的表示方法。

Prolog的自定义操作符看上去如下：

```
:- op(Precedence, Type, Name).
```

正如之前提及的，优先级是一个从0到1200的数字，数字越大，优先级越高（再次强调，这里的优先级高，是指函子使用在越外层，和我们平时理解的优先级高先运算和调用是相反的）。Type是一个原子，表示操作符的类型和结合性。比如+的这个原子是yfx，含义是说+是一个中缀操作符，f代表操作符，x和y代表参数。更进一步地说，x的优先级低于+的优先级，y的优先级低于或者等于+的优先级。这里有一些可能的type：

infix xfx, xfy, yfx
prefix fx, fy
suffix xf, yf

所以我们自定义的操作符is_dead代码如下：

```
?- op(500, xf, is_dead).
```

这里有一些内置操作符的定义。可以看到相同属性的操作符定义在一个子句中，通过最后一个参数给出名字的列表：

```
:- op(1200, xfx, [ :-, -->]).
:- op(1200,  fx, [ :-, ?-]).
:- op(1100, xfy, [;]).
:- op(1000, xfx, [ ',' ]).
:- op( 700, xfx, [ =, is, =.., ==, \==, =:=, =\=, <, >, =<,
>=]).
```

```
:- op( 500, yfx, [ +, -]).
:- op( 500,  fx, [ +, -]).
:- op( 300, xfx, [ mod ]).
:- op( 200, xfy, [ ^ ]).
```

最后需要明确的一点是，自定义操作符不会实现操作符的功能，而只是定义如果使用操作符。即，一个自定义的操作符不会包括查询在什么情况下为真的运算，它仅仅是Prolog在语法层面的扩充。所以如果操作符is_dead想上面那样定义，并且你直接查询：zed is_dead，Prolog不会报语法有错误，但是同时会证明的目标是：is_dead(zed)，这是Prolog内部的标准表示方法。所以这就是自定义操作符做的一切——只是将友好的语法转为Prolog真正内部的表示法。所以，Prolog将会如何回答这个查询呢？答案是false，因为Prolog试图证明：is_dead(zed)，但是在知识库中找不到能够匹配的目标。但是，如果我们扩展知识库如下：

```
:- op(500, xf, is_dead).

kill(marsellus, zed).
is_dead(X) :- kill(_, X).
```

这时Prolog会根据知识库的事实和规则，回答true。

分类: Prolog

标签: 操作符, 自定义操作符, 优先级, 结合性

« 上一篇: Learn Prolog Now 翻译 - 第九章 - 语句深究 - 第三节， 语句的检查
» 下一篇: Learn Prolog Now 翻译 - 第十章 - 中断和否定 - 第一节， 中断