All About Prolog Operators

I. Operator Declarations

a. Operators are functors written in a form (syntax) that makes some structures easier to read (syntactic sugar). For example, when we write \( x+y*z \), we call the ‘+’ and the ‘*’ operators. Internally, in Prolog, the expression \( x+y*z \) is represented as \( +(x,*(y,z)) \) (which is functional notation) which is awkward since we are accustomed to the other more familiar *infix notation*.

b. Operators do not “cause” any arithmetic to be carried out, that is, in Prolog \( 3+7 \) does not unify with 7. The term \( 3+4 \) is just another way to write \( +(3,4) \) which is a Prolog (a data) structure like say, \( p(a,b) \).

c. Three things are defined for each operator: (1) *position*, (2) *precedence* and (3) *associativity*. Operators written between their arguments are called *infix* operators, e.g., ‘+’, ‘-’, ‘*’, and ‘/’. Operators that come before their arguments are called *prefix*, e.g., as in arithmetic negation ‘-’.

   Operators that are written after their arguments are called *postfix*, e.g., as in “\( x! \)”.

d. *Precedence* tells us which operation is carried out first. For example, when we write \( x+y*z \) we assume that \( y*z \) is done before it is added to \( x \). The equivalent Prolog structure =\( +(x,*(y,z)) \) makes explicit that the multiplication is done before the addition. Precedence is an integer that is usually between 1 and 1023 (or 255) and is dependent on the Prolog interpreter you are using. The higher precedence has a value closer to 1.

e. *Association* comes into focus when we concatenate operators of the same precedence, e.g., \( 8/2/2 \). Does this mean \( 8/(2/2) \) or \( (8/2)/2 \)? A *left associative* operator must have the same or lower precedence on the left and lower precedence on the right. The arithmetic operators: ‘+’, ‘-’, ‘*’ and ‘/’ are all left associative, so we interpret “\( 5+(8/2/2) \)” as “\( 5+(8/(2/2)) \)”.

f. *Infix* operators are declared with the following templates: \( xfx \), \( xfy \), \( yfx \), and \( yfy \). The letters \( x \) and \( y \) are “pictures” of the possible associativity. Assuming there are no parentheses, \( y \) means that the argument can contain operators of the same or lower precedence than this operator. An \( x \) means that any operators in the argument must be strictly lower precedence than this operator. Thus, \( yfx \) means left associative and \( xfy \) means right associative. *Prefix* operators are declared with \( fx \) and \( fy \), respectively.

g. Operators are declared via commands: \( \text{?-op(Precedence Int, Association Pic, Operator Const)} \). Some examples include: \( \text{?-op(1200,fx, ’?’)} \), \( \text{?-op(1200,fx, ’:’) } \) , \( \text{?-op(1100,xfy, ’;’)} \), \( \text{?-op(1000,xfy, ’,’)} \), \( \text{?-op(700,xfx,’=..’)} \), \( \text{?-op(500,yfx,’+’) } \) , \( \text{?-op(400,yfx,’*’)} \), etc.

II. Examples

   ![Illustrative only—using neg on both sides of a :: is bad Prolog programming practice.]

| \( \text{?-op(200,fx,~).} \) | 1 | ?- neg(p,Z).  
\( \text{neg(\(-P,P\).)} \). | \( Z = \neg p \).  
\( \text{neg(P,\(-P\).)} \). | \( Z = \neg p \).  
| \( \text{?-op(200,fx,~).} \) | 2 | ?- neg(p,Z).  
\( \text{neg(\(-P,P\).)} \). | \( Z = p \).  
| \( \text{?-op(200,fx,~).} \) | 4 | ?- neg(\(-\neg(p)\),Z).  
\( \text{neg(\(-\neg(P),P1\).)} \). | \( Z = \neg p \).  
\( \text{neg(\(-\neg(P)),P).} \). | \( Z = p \).  
| \( \text{?-op(200,fx,~).} \) | 5 | ?- neg(\(-\neg(p)\),Z).  
\( \text{neg(\(-\neg(P),P1\).)} \). | \( Z = \neg p \).  
\( \text{neg(\(-\neg(P)),P).} \). | \( Z = p \).  
| \( \text{?-op(200,fx,~).} \) | 6 | ?- neg(\(-\neg(p)\),Z).  
\( \text{neg(\(-\neg(P),P1\).)} \). | \( Z = \neg p \).  
\( \text{neg(\(-\neg(P)),P).} \). | \( Z = p \).  

1