**左递归规则**

这是一个带有左递归表达式规则的ANTLR 4语法示例：

stat: expr '=' expr ';' // e.g., x=y; or x=f(x);

| expr ';' // e.g., f(x); or f(g(x));

;

expr: expr '\*' expr

| expr '+' expr

| expr '(' expr ')' // f(x)

| id

;

在直接上下文无关语法中，这样的规则是不明确的，因为1+2\*3它可以将任一运算符解释为首先发生，但ANTLR使用语义谓词重写非左递归和明确：

expr[int pr] : id

( {4 >= $pr}? '\*' expr[5]

| {3 >= $pr}? '+' expr[4]

| {2 >= $pr}? '(' expr[0] ')'

)\*

;

谓词通过比较当前运算符的优先级与前一运算符的优先级来解决歧义。expr [pr]的扩展只能匹配优先级满足或超过pr的子表达式。

**正式规则**

4.2版的正式4.0,4.1 ANTLR左递归消除规则被更改（简化），并在[ALL（\*）技术报告中](http://www.antlr.org/papers/allstar-techreport.pdf)进行了布局：

* 二进制表达式是包含规则的递归调用的表达式，作为备选的第一个和最后一个元素。
* 后缀表达式包含规则的递归调用作为替代的第一个元素，但不作为最后一个元素。
* 前缀表达式包含规则的递归调用作为替代的最后一个元素，但不作为第一个元素。

没有“三元”表达式这样的东西 - 它们只是伪装的二元表达式。

正确的关联性说明符曾经出现在单个令牌上，但无论如何都是在替代的基础上完成的，所以现在选择的是个别备选方案; 例如，

e : e '\*' e

| e '+' e

|<assoc=right> e '?' e ':' e

|<assoc=right> e '=' e

| INT

;

如果您的4.0或4.1语法使用右关联三元运算符，则需要更新语法以包含<assoc=right>在替代运算符上。为了平滑过渡，<assoc=right>仍然允许在令牌引用上但忽略它。