语义谓词{...}?是用目标语言编写的布尔表达式

## **做出预测的解析决策**

prog: {false}? 'return' INT ; // throws FailedPredicateException

ANTLR将{false}?语法转换为生成的解析器中的条件：

if ( !false ) throw new FailedPredicateException(...);

## **寻找可见的谓词**

解析器不会在操作或令牌引用之后发生的预测期间评估谓词。让我们首先考虑一下动作和谓词之间的关系。

在执行创建操作之前评估该谓词x将违反语法中隐含的操作顺序。

解析器在决定匹配哪个替代方案之前无法执行操作。例如，在以下规则中，解析器{java5}?在提交该替代项之前无法在谓词前执行操作。

@members {boolean allowgoto=false;}

stat: {System.out.println("goto"); allowgoto=true;} {java5}? 'goto' ID ';'

| ...

;

如果我们无法在预测期间执行操作，则不应评估{java5}?谓词，因为它取决于该操作。

预测过程也无法通过令牌引用来查看。

例如，在以下语法中，谓词期望getCurrentToken返回ID令牌。

stat: '{' decl '}'

| '{' stat '}'

;

decl: {istype(getCurrentToken().getText())}? ID ID ';' ;

expr: {isvar(getCurrentToken().getText())}? ID ;

预测过程忽略了不可见的谓词，将它们视为不存在。

## **使用依赖于上下文的谓词**

显然，我们只能在定义它们的规则中评估这些谓词。例如，下面的prog中的决定测试依赖于上下文的谓词是没有意义的{$i<=5}?。该$i局部变量甚至没有定义prog。

prog: vec5

| ...

;

vec5

locals [int i=1]

: ( {$i<=5}? INT {$i++;} )\* // match 5 INTs

;

例如，对stat中的else子规则的可选分支的预测“下降”为stat的结尾，并继续在调用prog规则中查找符号。

## **Lexer规则中的谓词**

例如，输入枚举将匹配ENUM规则和ID规则。如果枚举后的下一个字符是空格，则两个规则都不能继续。

ENUM : 'enum' ;

ID : [a-z]+ ;

另一方面，如果输入后的下一个字符enum是一个字母，那么只有ID可行。

作为一个例子，这是将枚举作为词法分析器中的关键字匹配的另一种方法：

ENUM: [a-z]+ {getText().equals("enum")}?

{System.out.println("enum!");}

;

ID : [a-z]+ {System.out.println("ID "+getText());} ;

打印操作ENUM最后显示，仅在当前输入匹配[a-z]+且谓词为真时才执行。让我们构建并测试Enum3它是否区分枚举和标识符：

$ antlr4 Enum3.g4

$ javac Enum3.java

$ grun Enum3代币

= > enum abc

= > EOF

< = enum ！

ID abc

这很好用，但它真的只是出于教学目的。将enum关键字与简单的规则匹配起来更容易理解和更有效：

ENUM : 'enum' ;