Lex的实现

09009331 赵成

**一、综述**

Lex是一种自动词法分析器，可以根据输入的规则来自动生成解析词法的程序源代码。这里用实现了C++实现了一个简单的Lex，主要包括下面几个部分：

1. 正则表达式词法分析器
2. 正则表达式解析
3. 正则表达式向DFA转换
4. 由DFA自动生成词法解析程序
5. Lex规则文件解析器

**二、正则表达式词法分析器**

该部分代码在reg\_lexer.h与reg\_lexer.cc中。

这里实现的正则表达式的词法单元，包含下面这几个部分：

1. 引用已有定义，如{number}
2. 集合，如[a-z]。其中，如果存在“-”符号，就以两边元素为左右边界，把之间全部元素加入集合中，对于其他字符直接加入集合中。比如[0-9A]就包含0、1、2、3、4、5、6、7、8、9、A
3. 字符串，如”abcd”，引号内的内容作为整体看待，不进行正则表达式解析
4. 特殊符号，包括\*、|、+、？、（、）和$
5. 字符，除特殊符号外的所有ASCII字符。对于“\”开头的字符进行转义，转义风格与C一致

正则表达式词法分析器的实现采用手写进行字符串匹配的方式实现，比较简单。每个词法单元用Token类型表示。

**三、正则表达式解析**

该部分代码在reg\_parser.h与reg\_parser.cc中。

实现的正则表达式文法表示为

E → C

E → E | C

C → S

C → SC

S → F

S → F+

S → F\*

S → F?

F → (E)

S → 定义名

S → 集合

S → 特殊符号

S → 字符

解析方式采用递归向下，因为正则表达式递归层次并不深，所以问题不大。正则表达式的叶子节点全部都是无特殊含义的字符，因此全部按照出现顺序放在positions数组中。

**四、正则表达式向DFA转换**

转换算法采用了龙书上179页的Algorithm 3.36，由正则表达式直接向DFA转换，直接省去了向NFA转换的中间过程，一方面大大加快了速度，另一方面减少了生成的DFA的状态数目。

1. **正则表达式的节点**

正则表达式以词法单元作为节点，当成树进行解析，节点类型为RegNode，包含下面几个元素：firstpos、lastpos、nullable，对应于龙书上175页的三个函数。firstpos和lastpos分别表示该节点可能出现的第一个或者最后一个字符，nullable表示该节点是否能表示成ε。

在递归解析正则表达式时，我们可以由下面的规则计算出每个节点的firstpos、lastpos和nullable：

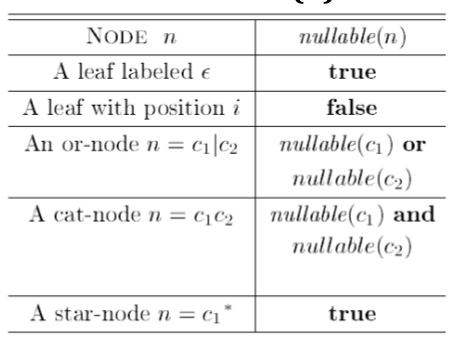


图 4.1 计算nullable

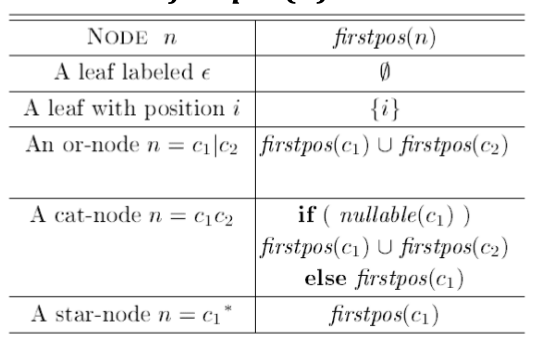


图 4.2 计算firstpos

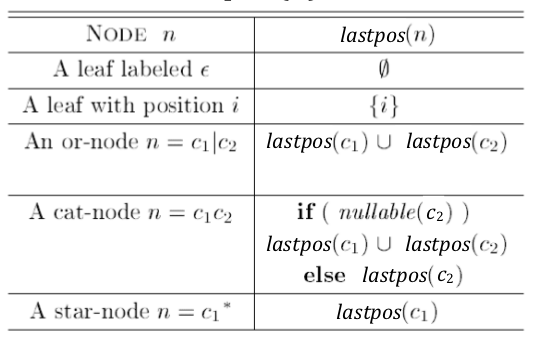


图 4.3 计算lastpos

1. **由firstpos、lastpos和nullable计算出followpos**

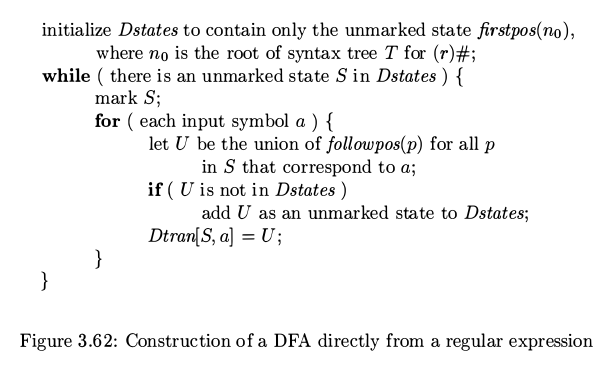
followpos表示了正则表达式语法树中，每一个叶子节点后可能出现的字符。只有在下面两种情况下，一个字符后可能会跟随其他字符：

1. c1 c2：会出现lastpos(c1)被firstpos(c2)跟随，则有对c1的每一个字符a，都有followpos(a) = followpos(a) ∪ firstpos(c2)
2. c\*：会出现lastpos(c)被firstpos(c)跟随，则有对1的每一个字符a，都有followpos(a) = followpos(a) ∪ firstpos(c)

计算followpos的过程可以在解析正则表达式时完成。这里的实现中，节点连接和节点闭包有node\_cat和node\_closure完成，解析可以先完成firstpos、lastpos、nullable的计算，然后再计算出followpos。

1. **根据followpos得到DStates和DTran**

转换成DFA时，正则表达式语法树的根的fristpos构成第一个DState，然后根据Figure 3.62中的过程构建出DTran和其余的DState，得到一个完整的DFA。



1. **由正则表达式规则构建语法树**

输入的正则表达式规则分为两种，一种是在定义区预先定义好的规则，另一种是在规则区定义的需要触发动作的规则。

前者在构建时不加入终结符$，构建完成之后把名称和正则表达式语法树根节点保存在表中，之后解析正则表达式时遇到{定义名}就直接从表中取出节点。

后者在构建出语法树后，再手动在其后连接一个终结符，用Token(FINISH\_CHAR);表示。这样在生成DFA时可以标记出终结状态。

1. **多个正则表达式的处理**

在输入规则时，绝大部分时候都会输入不止一条规则，也就需要处理不止一棵正则表达式语法树。对此这里的处理方法是每处理一条规则就把得到的根节点保存起来，假设为C1，然后再和原来的保存根节点合并起来，假设为C2，那么新得到的根节点就是C1 | C2，这样可以保证最后得到一棵完整的正则表达式语法树，且能生成完整的DFA。

**五、由DFA自动生成词法解析程序**

首先，需要把DTran打印到文件里，打印形式为一个二维数组，第一维是DState的id，第二维是全部的ASCII字符。如果状态i由字符a跳转到状态j，那么DTran[i][‘a’] = j。如果某一条路径不存在，就把DTran相应的元素置为-1。

然后，根据终结符$出现的位置，找出所有的终结状态，然后用一个数组FinishedStates标记出哪些状态是终结状态，输出这个数组，并且生成相应的动作。

最后输出一个范例函数yylex()，用于自动读入字符、进行状态跳转。yylex函数需要保证匹配最长的字符串，并且在yytext和yyleng变量中保存当前匹配的字符串内容和字符串长度。

**六、Lex规则文件解析器**

Lex规则分为三个区域，定义区、规则区和程序区。在lex.cc和lex.h中定义了解析Lex规则的程序，然后通过基本的字符串比较来区分出正则表达式规则，再由正则表达式解析器生成相应的语法树和DFA。

**七、综述**

这里使用了优化过后的由正则表达式直接生成DFA的算法，并且补充了龙书中并未提到的部分内容，比如lastpos的计算、followpos的计算和多个正则表达式的处理，比较好地消化理解了龙书中词法分析的内容。