第3章 词法分析

词法分析器作用

- 词法分析是读入源程序的输入字符、将它们组成词素，生成并输出一个词法单元序列，每个词法单元对应于一个词素
- 常见的做法
  - 由语法分析器调用，需要的时候不断读取、生成词法单元
  - 可以避免额外的输入输出
- 在识别出词法单元之外，还会完成一些不需要生成词法单元的简单处理，比如删除注释、将多个连续的空白字符压缩成一个字符等

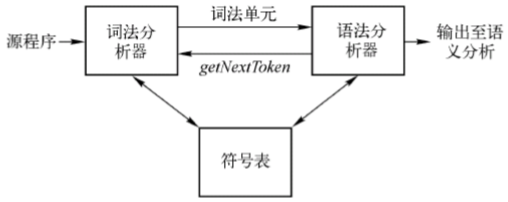


图 3-1　词法分析器与语法分析器之间的交互

在识别出词法单元之外，还会完成一些不需要生成词法单元的简单处理，比如删除注释、将多个连续的空白字符压缩成一个字符等，另一个任务是将编译器生成的错误消息和源程序的位置联系起来。

相关概念

- 词素（Lexeme）
  - 源程序中的字符序列，它和某类词法单元的模式匹配，被词法分析器识别为该词法单元的实例。
- 词法单元（Token）：
  - 包含单元名（Token-name）和可选的属性值(attribute-value)
  - 单元名是表示某种词法单位抽象符号。语法分析器通过单元名即可确定词法单元序列的结构。

模式：词法单元对应的词素可能具有的形式，可以用正则表达式来表示

一个模式匹配多个词素时，必须通过属性来传递附加的信息。属性值将被用于语义分析、代码生成等阶段。不同的目的需要不同的属性。因此，属性值通常是一个结构化数据。词法单元id的属性：词素、类型、第一次出现的位置

正则表达式

字母表、串、语言、前缀、后缀、子串。运算的优先级：* > 连接符 > |

- 基本运算符：并 连接 闭包
- 扩展运算符
  - 一个或多个：r^+，等价于rr*
  - 零个或一个： r?，等价于ε |r
  - 字符类 [abc]等价于a|b|c, [a-z]等价于a|b|…|z

表达式	匹配	例子
c	单个非运算符字符 c	a
\c	字符 c 的字面值	*
“s”	串 s 的字面值	“**”
.	除换行以外的任何字符	a.*b
^	一行的开始	^abc
$	行的结尾	abc$
[s]	字符串 s 中的任何一个字符	[abc]
[^s]	不在串 s 中的任何一个字符	[^abc]
r*	由和 r 匹配的零个或多个串连接成的串	a*
r^+	由和 r 匹配的一个或多个串连接成的串	a+
r?	零个或一个 r	a?
r{m,n}	最少 m 个，最多 n 个 r 的连接	a{1,5}
r_1r_2	r_1后加上r_2	ab
$r_1 \mid r_2$	r_1或r_2	a|b
(r)	与 r 相同	(a|b)
r_1/r_2	后面跟有r_2时的r_1	abc/123

词法单元的识别

状态转换图

状态(state)：表示在识别词素的过程中可能出现的情况

状态看作是已处理部分的总结

某些状态为**接受状态或最终状态**，表明已经找到词素

加上*的接受状态表示最后读入的符号不在词素中

开始状态（初始状态）：用start边表示

边(edge)：从一个状态指向另一个状态；边的标号是一个或者多个符号

如果当前符号为s，下一个输入符号为a，就沿着从s离开，标号为a的边到达下一个状态

有穷自动机

- NFA: 一个符号标记离开同一状态的多条边
- DFA: 对于每个状态和字母表中的每个字符，有且仅有一条离开该状态、以该符合为标号的边

- NFA: 可以有边的标号是ε
- DFA: 没有标记为ε的边

只要存在从开始状态到接受状态的路径，符号串就认为被NFA接受。

NFA到DFA

- 输入：一个NFA *N*
- 输出：一个接受相同语言的DFA *D*

s表示N中的单个状态，T代表N的一个状态集

操作	描述
ϵ-*closure*(s)	能够从NFA的状态 s 开始只通过ϵ 转换到达的 NFA状态集合
ϵ-*closure*(T)	能够从T中某个NFA状态 s 开始只通过ϵ 转换到达的NFA状态集合，即$\cup_{s \in T} \epsilon$-*closure*(s).
move(T, a)	能够从 T 中某个状态 s 出发通过标号为a的转换到达的NFA状态的集合

图 3-31　NFA 状态集上的操作

- D的开始状态是ε-closure(s_0)，D的接受状态是所有至少包含了N的一个接受状态的状态集合。

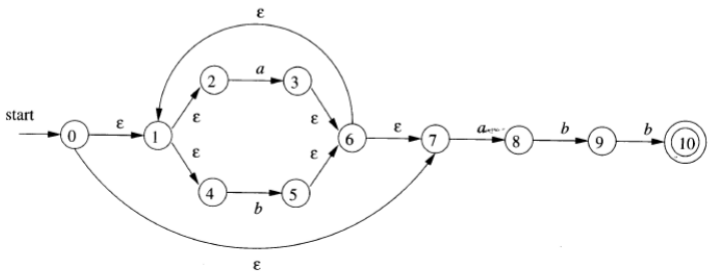
```
一开始, ε-closure(s0)是 Dstates 中的唯一状态，且它未加标记；
while (在 Dstates 中有一个未标记状态T) {
        给T加上标记;
        for (每个输入符号 a) {
                U = ε-closure(move(T, a));
                if (U 不在Dstates中 )
                        将U 加入到Dstates中，且不加标记；
                Dtran[T, a] = U;
        }
}
```

图 3-32　子集构造法

示例：

NFA到DFA转换的示例

- A：=ε-closure(0)={0,1,2,4,7}
- B：Dtran[A,a]= ε -closure(move(A,a))= ε - closure({3,8})={1,2,3,4,6,7,8}
- C：Dtran[A,b]= ε -closure(move(A,b))= ε -closure({5})={1,2,4,5,6,7}
- D：Dtran[B,b]=ε -closure(move(B,b))= {1,2,4,5,6,7,9}
- ……
- E:　Dtran[D,b] = ε -closure(move(B,b))= {1,2,4,5,6,7,10}



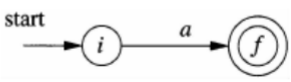
正则表达式到NFA

- 输入：字母表Σ上的一个正则表达式*r*
- 输出：一个接受*L(r)*的NFA *N*
- 基本思想
  - 根据正则表达式的递归定义，按照正则表达式的结构递归地构造出相应的NFA
  - 算法分成两个部分：
    - 基本规则处理ε和单符号的情况
    - 对于每个正则表达式的运算，建立构造相应NFA的方法

递归构造：

- 基本规则：
  - 表达式ε，

  - 表达式a，



- 归纳规则
  - 正则表达式*s*和*t*的NFA分别是 *N(s)*和*N(t)*
  - *r=s|r*， *r*的NFA *N(r)*

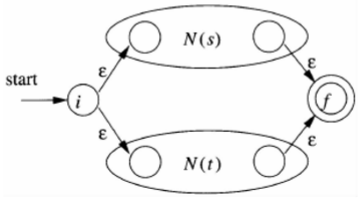


图 3-40　两个正则表达式的并的 NFA

- 归纳规则
  - 正则表达式*r=st*， *N(r)*

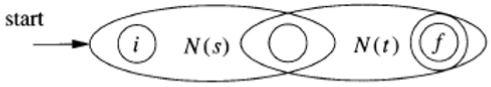


图 3-41　两个正则表达式的连接的 NFA

- 归纳规则
  - 正则表达式 *r=s*, N(r)*

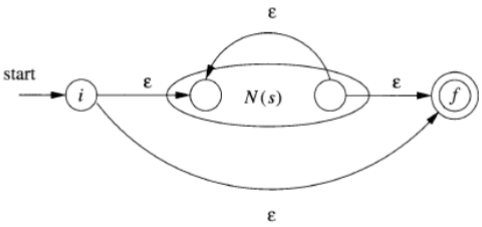


图 3-42　一个正则表达式的闭包的 NFA

DFA状态最小化

不考