

# 词法分析 (2)

魏恒峰

hfwei@nju.edu.cn

2021 年 11 月 09 日 (周二)





## 手写词法分析器

识别字符串  $s$  中符合某种词法单元模式的所有词素

```
if ab42>=3.14
    xyz :=2.99792458E8
else
    xyz:= 2.718
    abc := 1024
```

ws if else id integer real sci relop assign (:=)

识别字符串  $s$  中符合某种词法单元模式的所有词素

```
if ab42>=3.14
    xyz :=2.99792458E8
else
    xyz:= 2.718
    abc := 1024
```

ws if else id integer real sci relop assign (:=)

识别字符串  $s$  中符合某种词法单元模式的前缀词素

识别字符串  $s$  中符合某种词法单元模式的所有词素

```
if ab42>=3.14
    xyz :=2.99792458E8
else
    xyz:= 2.718
    abc := 1024
```

ws if else id integer real sci relop assign (:=)

识别字符串  $s$  中符合某种词法单元模式的前缀词素

识别字符串  $s$  中符合特定词法单元模式的前缀词素

识别字符串  $s$  中符合**特定词法单元模式**的**前缀词素**

**分支:** 先判断属于哪一类, 然后进入特定词法单元的前缀词素匹配流程

识别字符串  $s$  中符合**某种词法单元模式**的**前缀词素**

**循环:** 返回当前识别出来的词法单元与词素, 继续识别下一个前缀词素

识别字符串  $s$  中符合**某种词法单元模式**的**所有词素**

先: `ws if else id integer`

然后: `relop`

最后: `real sci`

留给大家: `assign (:=)`

```
4)    public int line = 1;
5)    private char peek = ' ';
6)    private Hashtable words = new Hashtable();
```

line: 行号, 用于调试

peek: 下一个向前看字符 (Lookahead)

words: 从词素到词法单元标识符或关键词的映射表

```
words.put("if", if)
words.put("else", else)
```



识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

**ws:** blank   tab   newline

识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

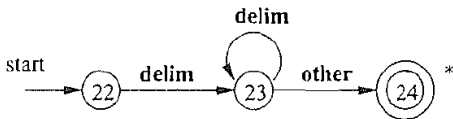
**ws: blank tab newline**

```
12)    public Token scan() throws IOException {  
13)        for( ; ; peek = (char)System.in.read() ) {  
14)            if( peek == ' ' || peek == '\t' ) continue;  
15)            else if( peek == '\n' ) line = line + 1;  
16)            else break;  
17)        }
```

识别空白部分, 并忽略之

识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

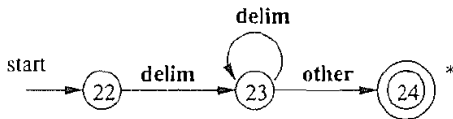
ws: blank tab newline



用于识别**空白符**的状态转移图

识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

ws: blank tab newline

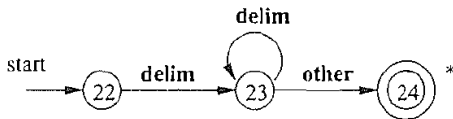


用于识别空白符的状态转移图

\*: 识别出的空白符**不包含**当前 peek 指向的字符

识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

ws: blank tab newline



用于识别空白符的状态转移图

\*: 识别出的空白符**不包含**当前 peek 指向的字符

22: 碰到 other 怎么办?

识别字符串  $s$  中符合**特定词法单元模式**的**前缀词素**

**integer**: 整数 (简化处理, 允许以 0 开头)

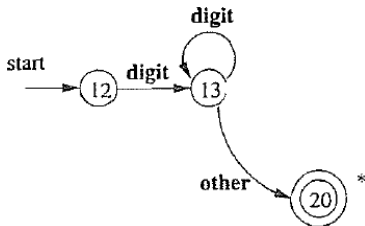
识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

**integer**: 整数 (简化处理, 允许以 0 开头)

```
18)         if( Character.isDigit(peek) ) {  
19)             int v = 0;  
20)             do {  
21)                 v = 10*v + Character.digit(peek, 10);  
22)                 peek = (char)System.in.read();  
23)             } while( Character.isDigit(peek) );  
24)             return new Num(v);  
25)         }
```

识别字符串  $s$  中符合**特定词法单元模式**的**前缀词素**

**integer**: 整数 (简化处理, 允许以 0 开头)

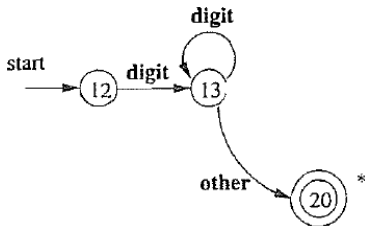


用于识别**integer**的状态转移图



识别字符串  $s$  中符合**特定词法单元模式**的**前缀词素**

**integer**: 整数 (简化处理, 允许以 0 开头)



用于识别**integer**的状态转移图

**12**: 碰到 other 怎么办?

识别字符串  $s$  中符合**特定词法单元模式**的**前缀词素**

**id**: 字母开头的字母/数字串

识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

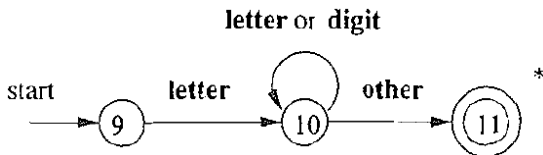
**id**: 字母开头的字母/数字串

```
26)      if( Character.isLetter(peek) ) {
27)          StringBuffer b = new StringBuffer();
28)          do {
29)              b.append(peek);
30)              peek = (char)System.in.read();
31)          } while( Character.isLetterOrDigit(peek) );
32)          String s = b.toString();
33)          Word w = (Word)words.get(s);
34)          if( w != null ) return w;
35)          w = new Word(Tag.ID, s);
36)          words.put(s, w);
37)          return w;
38)      }
```

**识别词素、判断是否是预留的关键字或已识别的标识符、保存该标识符**

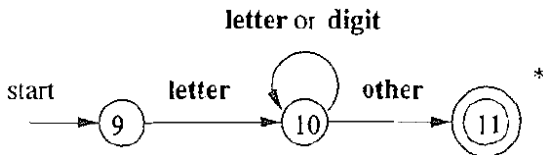
识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

id: 字母开头的字母/数字串



识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

id: 字母开头的字母/数字串



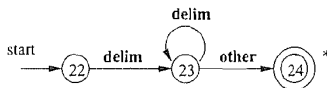
9: 碰到 other 怎么办?

识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

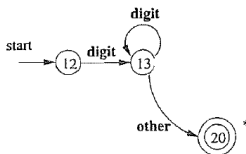
```
39)      Token t = new Token(peek);  
40)      peek = ' ';  
41)      return t;
```

错误处理模块: 出现**词法错误**, 直接报告异常字符

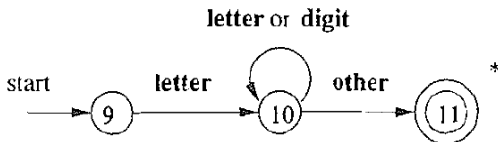
识别字符串  $s$  中符合**某种词法单元模式**的前缀词素 (SCAN())



ws: 空白符



integer: 整数

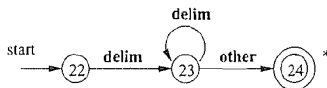


id: 标识符

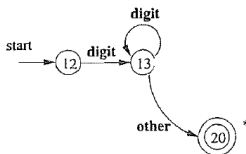
```
39) Token t = new Token(peek);
40) peek = ' ';
41) return t;
```

错误处理模块

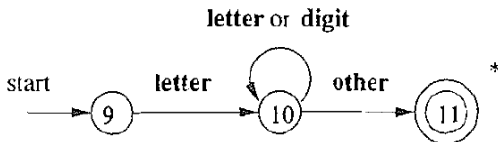
识别字符串  $s$  中符合**某种词法单元模式**的前缀词素 (SCAN())



ws: 空白符



integer: 整数



id: 标识符

```
39)      Token t = new Token(peek);
40)      peek = ' ';
41)      return t;
```

错误处理模块

**关键点:** 合并 22, 12, 9, 根据**下一个字符**即可判定词法单元的类型

否则, 调用错误处理模块 (对应 other), 报告**该字符有误**, 并忽略该字符



```
1) package lexer;                                // 文件 Lexer.java
2) import java.io.*; import java.util.*;
3) public class Lexer {
4)     public int line = 1;
5)     private char peek = ' ';
6)     private Hashtable words = new Hashtable();
7)     void reserve(Word t) { words.put(t.lexeme, t); }
8)     public Lexer() {
9)         reserve( new Word(Tag.TRUE, "true") );
10)        reserve( new Word(Tag.FALSE, "false") );
11)    }
12)    public Token scan() throws IOException {
13)        for( ; ; peek = (char)System.in.read() ) {
14)            if( peek == ' ' || peek == '\t' ) continue;
15)            else if( peek == '\n' ) line = line + 1;
16)            else break;
17)        }
```

```

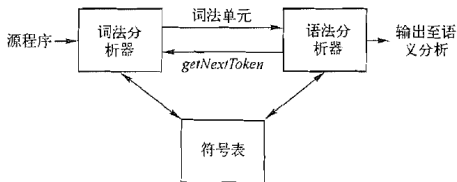
18)         if( Character.isDigit(peek) ) {
19)             int v = 0;
20)             do {
21)                 v = 10*v + Character.digit(peek, 10);
22)                 peek = (char)System.in.read();
23)             } while( Character.isDigit(peek) );
24)             return new Num(v);
25)         }
26)         if( Character.isLetter(peek) ) {
27)             StringBuffer b = new StringBuffer();
28)             do {
29)                 b.append(peek);
30)                 peek = (char)System.in.read();
31)             } while( Character.isLetterOrDigit(peek) );
32)             String s = b.toString();
33)             Word w = (Word)words.get(s);
34)             if( w != null ) return w;
35)             w = new Word(Tag.ID, s);
36)             words.put(s, w);
37)             return w;
38)         }
39)         Token t = new Token(peek);
40)         peek = ' ';
41)         return t;
42)     }
43) }

```

识别字符串  $s$  中符合某种词法单元模式的所有词素

外层循环调用 `SCAN()`

或者, 由语法分析器按需调用 `SCAN()`



考虑例子 “123abc”

考虑例子 “123abc”

$\langle \text{num}, 123 \rangle$        $\langle \text{id}, abc \rangle$

考虑例子 “123abc”

$\langle \text{num}, 123 \rangle$        $\langle \text{id}, abc \rangle$

交给语法分析器报告语法错误

识别字符串  $s$  中符合**特定词法单元模式**的**前缀词素**

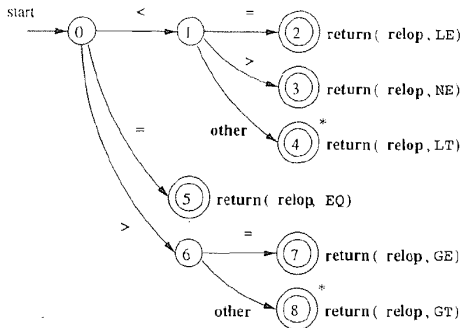
**relop**<sup>1</sup>: <   >   <=   >=   =   <>

---

<sup>1</sup>此处,= 是判断是否相等的关系运算符。请考虑, 如果 = 表示赋值, == 表示相等判断, 该如何设计词法分析器?

## 识别字符串 $s$ 中符合**特定词法单元模式**的前缀词素

**relop**<sup>1</sup>: < > <= >= = <>

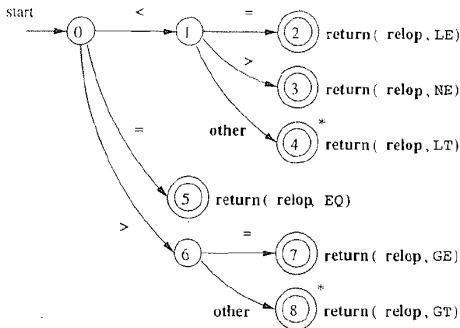


<sup>1</sup>此处, = 是判断是否相等的关系运算符。请考虑, 如果 = 表示赋值, == 表示相等判断, 该如何设计词法分析器?



识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

**relop**<sup>1</sup>: < > <= >= = <>

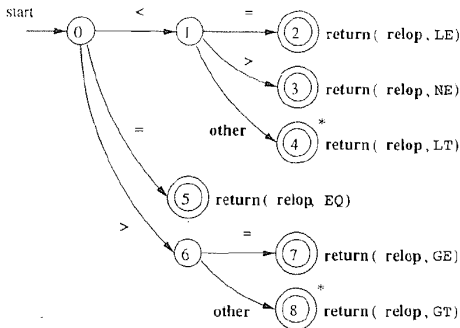


**“最长优先原则”**: 例如, 识别出 <=, 而不是 < 与 =

<sup>1</sup>此处, = 是判断是否相等的关系运算符。请考虑, 如果 = 表示赋值, == 表示相等判断, 该如何设计词法分析器?

识别字符串  $s$  中符合**特定词法单元模式**的前缀词素

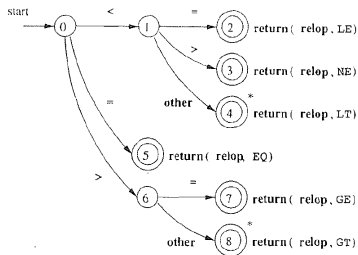
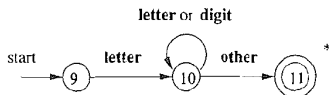
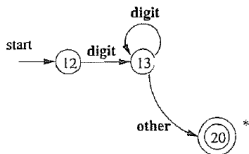
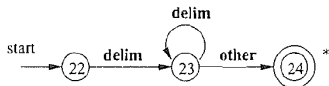
**relop**<sup>1</sup>: < > <= >= = <>



**“最长优先原则”**: 例如, 识别出 <=, 而不是 < 与 =  
**0**: 碰到 other 怎么办?

<sup>1</sup>此处, = 是判断是否相等的关系运算符。请考虑, 如果 = 表示赋值, == 表示相等判断, 该如何设计词法分析器?

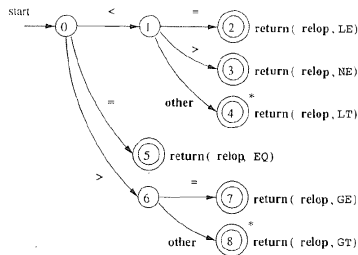
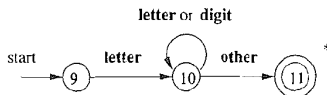
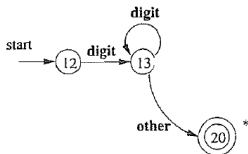
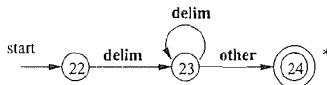
识别字符串  $s$  中符合**某种词法单元模式**的**前缀词素** (SCAN())



```

39) Token t = new Token(peek);
40) peek = ' ';
41) return t;
  
```

识别字符串  $s$  中符合**某种词法单元模式**的**前缀词素** (SCAN())



```
39) Token t = new Token(peek);
40) peek = ' ';
41) return t;
```

**关键点:** 合并 22, 12, 9, 0, 根据**下一个字符**即可判定词法单元的类型

否则, 调用错误处理模块 (对应 other), 报告**该字符有误**, 并忽略该字符

但是, 词法分析器的设计并没有这么容易



ws if else id integer relop

根据**下一个字符**即可判定词法单元的类型

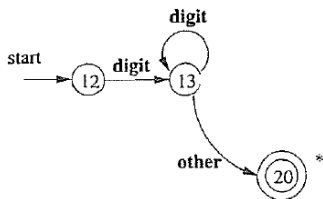
每个状态转移图的每个状态要么是**接受状态**, 要么带有**other 边**

ws if else id integer relop

根据**下一个字符**即可判定词法单元的类型

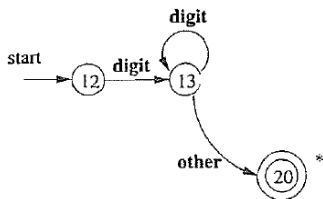
每个状态转移图的每个状态要么是**接受状态**, 要么带有**other 边**

如何同时识别 **integer**、**real** 与 **sci**?

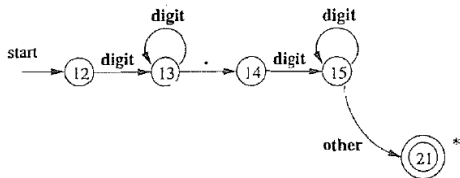


integer: 整数

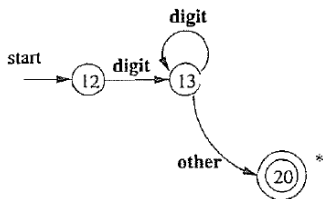




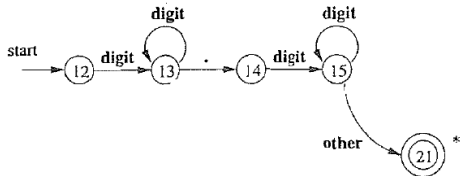
**integer:** 整数



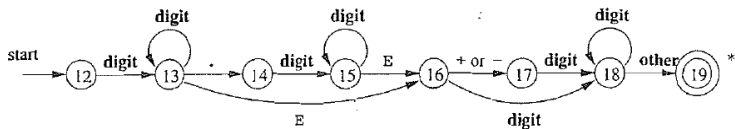
**real:** 浮点数 (无科学计数法)  
(不识别 2.)



**integer:** 整数



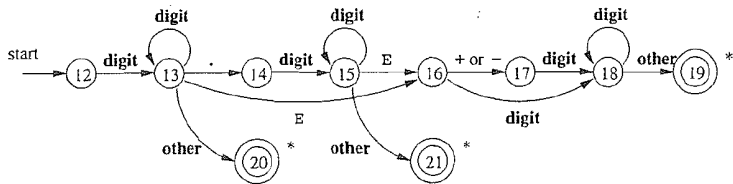
**real:** 浮点数 (无科学计数法)  
(不识别 2.)



**sci:** 带科学计数法的浮点数  
(2.99792458E8 3E8)

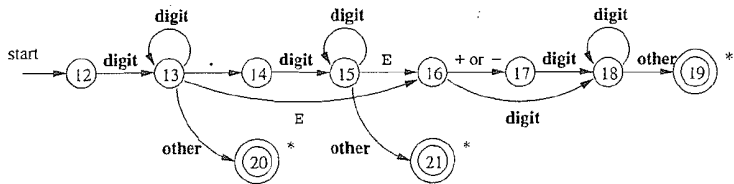
**num:** 整数部分[. 可选的小数部分][E[可选的 +-] 可选的指数部分]

num: 整数部分[. 可选的小数部分][E[可选的 +-] 可选的指数部分]



19, 20, 21 : 代表了不同的数字类型

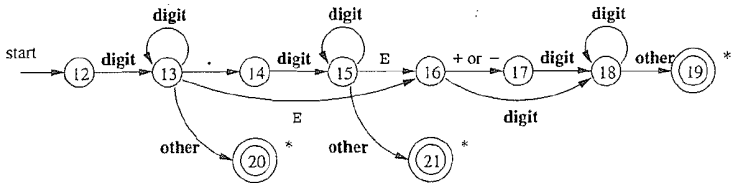
num: 整数部分[. 可选的小数部分][E[可选的 +-] 可选的指数部分]



19, 20, 21 : 代表了不同的数字类型

12 : 碰到 other 怎么办?

num: 整数部分[. 可选的小数部分][E[可选的 +-] 可选的指数部分]

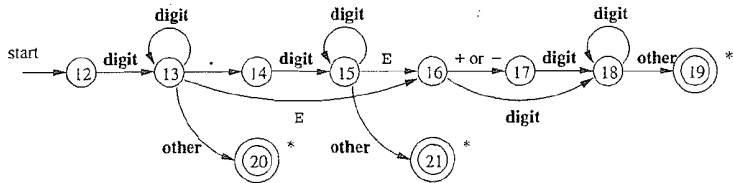


19, 20, 21 : 代表了不同的数字类型

12 : 碰到 other 怎么办?

(尝试其它词法单元或进入错误处理模块)

num: 整数部分[. 可选的小数部分][E[可选的 +-] 可选的指数部分]

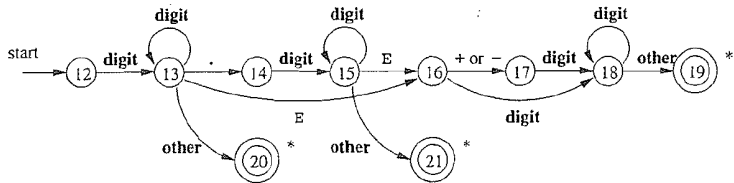


19, 20, 21 : 代表了不同的数字类型

12 : 碰到 other 怎么办? (尝试其它词法单元或进入错误处理模块)

14, 16, 17 : 碰到 other 怎么办?

num: 整数部分[. 可选的小数部分][E[可选的 +-] 可选的指数部分]



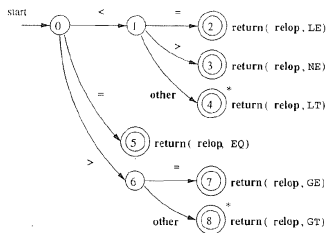
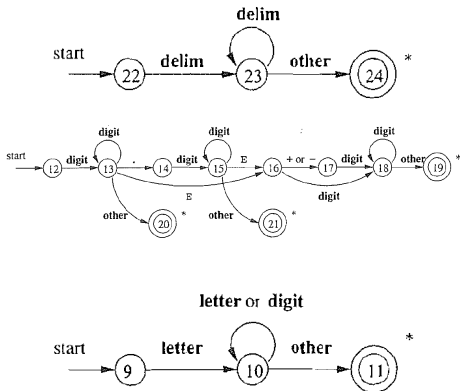
19, 20, 21 : 代表了不同的数字类型

12 : 碰到 other 怎么办? (尝试其它词法单元或进入错误处理模块)

14, 16, 17 : 碰到 other 怎么办? (回退, 寻找**最长匹配**)



# 识别字符串 $s$ 中符合某种词法单元模式的前缀词素 (SCAN())

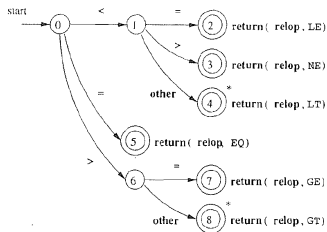
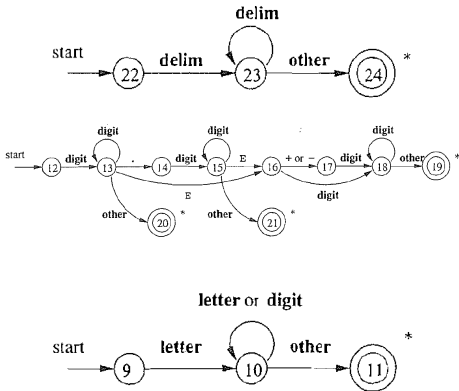


```

39) Token t = new Token(peek);
40) peek = ' ';
41) return t;

```

识别字符串  $s$  中符合**某种词法单元模式**的**前缀词素** (SCAN())



```

39) Token t = new Token(peek);
40) peek = ' ';
41) return t;

```

**关键点:** 合并 22, 12, 9, 0, 根据**下一个字符**即可判定词法单元的类型  
 否则, 调用错误处理模块 (对应 other), 报告**该字符有误**, 忽略该字符。

注意, 在sci中, 有时需要**回退**, 寻找最长匹配。

3) 有一个更好的方法,也是我们将在下面各节中采用的方法,就是将所有的状态转换图合并为一个图。我们允许合并后的状态转换图尽量读取输入,直到不存在下一个状态为止;然后像上面的2中讨论的那样取最长的和某个模式匹配的最长词素。

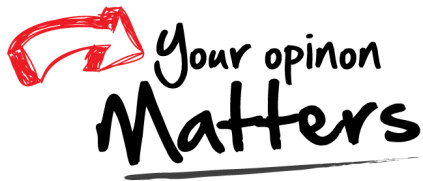
1.2345E+a

3) 有一个更好的方法，也是我们将在下面各节中采用的方法，就是将所有的状态转换图合并为一个图。我们允许合并后的状态转换图尽量读取输入，直到不存在下一个状态为止；然后像上面的 2 中讨论的那样取最长的和某个模式匹配的最长词素。

1.2345E+a

1.2345          E          +          a

Thank  
You!



Office 926

hfwei@nju.edu.cn