基于 FA 的词法分析程序实验报告

窦妍　141250030

南京大学软件学院

1.实验目的

编写一个词法分析程序，基于有限自动机原理，自己设计正则表达式（参考 java 语言的部分词法结构）并转化为 DFA，编写代码对语句进行词法分析。

2.实验描述

2.1 实验思路

1）针对要识别的单词写出正则表达式
2）构造每个正则表达式对应的 NFA
3）合并所有 NFA 并化简为 DFA
4）编写代码

2.2 程序代码实现

本程序使用 java 编写。读取文本文件 input.txt 对其中的语句进行分析：程序中以 State 记录当前状态，一开始处在初始状态；逐个读入字符后，先通过 SymbolAnalyzer 类识别字符属于字母、数字等标记，然后根据不同的标记参照状态机转换设计进入不同的 State，直到读入某一字符后进入一个终止状态，此时即完成一个 token 的分析，使用 token 类将分割出的字符串与词法分析结果输入，再接着读入下一字符直到分析完整个文件。将结果打印并存在 output.txt 里。

2.3 词法分析结果

本程序可识别出标识符、整数、浮点数、分隔符、操作符（包含一位与两位操作符）、字符串、注释、保留字等。

错误识别：程序以非分隔符结尾、错误的操作符!、错误的小数表示等。

3.正则表达式与 DFA

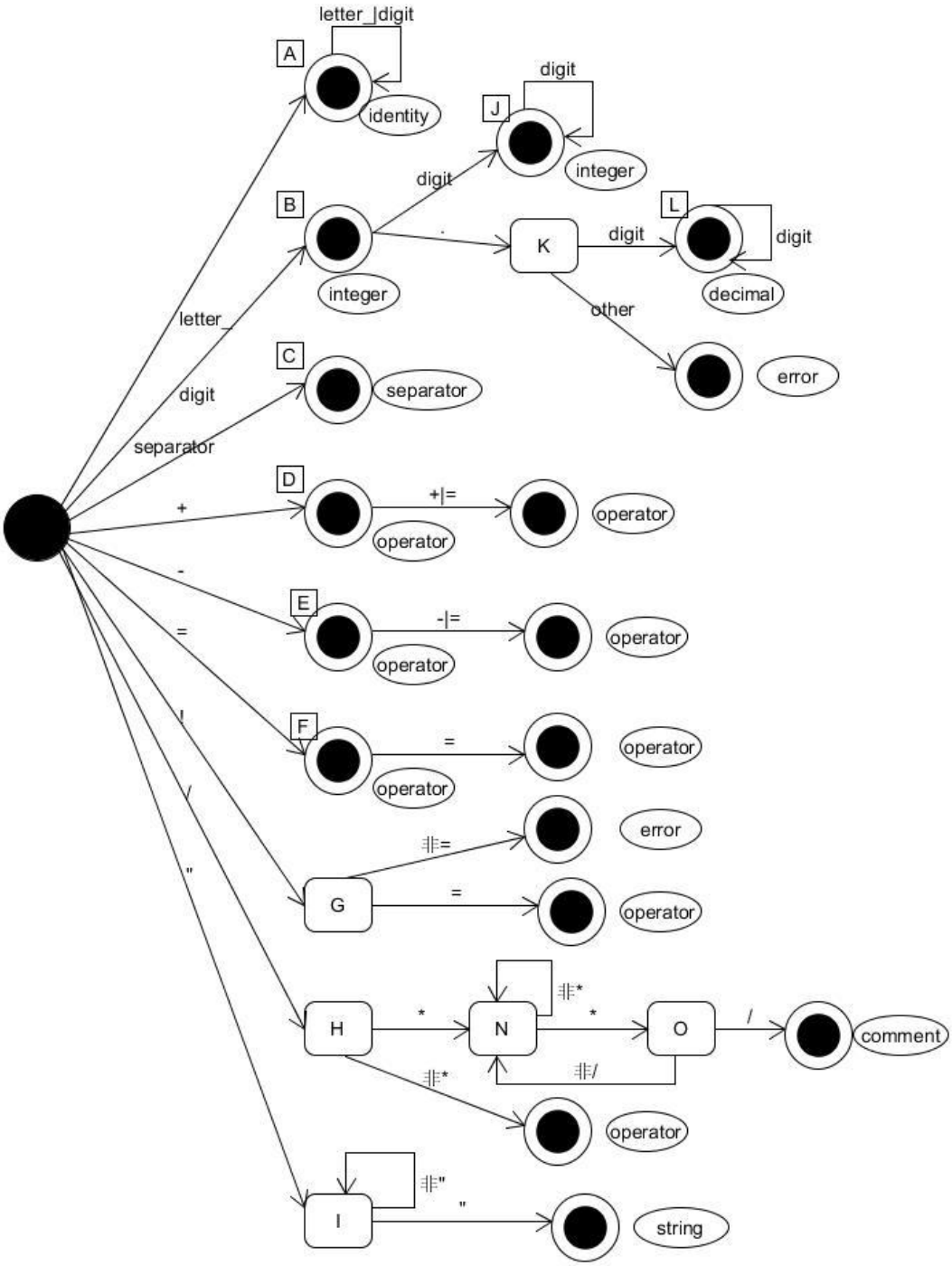
3.1 主要正则表达式

letter_ -> A | B | ... | Z | a | b | ... | z | _
digit -> 0 | 1 | ... | 9
separator ->{ | } | (|) | , | . | ; | : | [|] |”
operator -> + | - | * | / | = | < | > | &| += | -= | <= | >=| ! =

identifier -> letter_(letter_|digit)*
integer ->digit(digit)*
decimal ->(digit).(digit)(digit)*
String ->”(^”)*”
Comments -> /*([^*]*|*+[^/]+)*/*

3.2 DFA 转换图：

用大写字母 A-O 定义了各个状态，与 Java 代码中的状态定义一致。



4.程序输入与输出示例

Input.txt

```
public class Test{
   private void gcd(int u, int v){
```

```
       String s = "start";
      if(v!=0)
         u=v;
      else if(v==0)
         u=u-u/v*v;
      else if(v!1)
         u=1;
      int c=100;
      /*123*/
      while(c<0){
         if(c>=0){
            c=c-1;
         }
      }
   }
}
end
```

output.txt

```
reserved      public
reserved     class
identifier    Test
separator     {
reserved     private
reserved     void
identifier    gcd
separator     (
reserved     int
identifier    u
separator     ,
reserved     int
identifier    v
separator     )
separator     {
identifier    String
identifier    s
operator     =
string    "start"
separator     ;
reserved     if
separator     (
identifier    v
operator     !=
integer      0
separator     )
```

```
identifier    u
operator      =
identifier    v
separator     ;
reserved      else
reserved      if
separator     (
identifier    v
operator      ==
integer       0
separator     )
identifier    u
operator      =
identifier    u
operator      -
identifier    u
error     /
identifier    v
identifier    v
separator     ;
reserved      else
reserved      if
separator     (
identifier    v
error     !
integer       1
separator     )
identifier    u
operator      =
integer       1
separator     ;
reserved      int
identifier    c
operator      =
integer       100
separator     ;
comment       /*123*/
reserved      while
separator     (
identifier    c
integer       0
separator     )
separator     {
reserved      if
```

```
separator     (
identifier    c
operator      =
integer       0
separator     )
separator     {
identifier    c
operator      =
identifier    c
operator      -
integer       1
separator     ;
separator     }
separator     }
separator     }
separator     }
error     end
```

5.主要数据结构

Symbol：表示需要识别的所有标记符号（即处理后的读入的单个字符）。

```
enum Symbol {
        letter_,
        digit,
        separator,
        operator,
        add,
        minus,
        equal,
        dot,
        slash,
        quote,
        exclamation,
        star,
        none;
}
```

State:所有状态。

```
interface State {
        public State next(Token token);
}
```

OriginState:初始状态

```
class OriginState implements State
```

FinalState:终止状态

```
enum FinalState implements State{
    identifier,
    reserved,
    integer,
    decimal,
    separator,
    operator,
    comment,
    string,
    accept,
    error;
    @Override
    public State next(Token token) {
        return null;
    }

}
```

States: 所有非终结状态，并定义了状态之间的转换。

```
enum States implements State {

    A{
        @Override
        public State next(Token token) {
            if(token.hasNext()){
                switch(token.next()) {
                    case letter_:
                        return A;
                    case digit:
                        return A;
                    default:
                        token.back();
                        for(String word:reservedWord) {
                            if (token.getWord().equals(word))
                                return FinalState.reserved;
                        }
                        return FinalState.identifier;
                }
            }
            return FinalState.error;//can't end the code with a letter or digit
        }
    },
```

```
    B{
        @Override
        public State next(Token token) {
            if(token.hasNext()){
                switch (token.next()) {
                    case digit:
                        return J;
                    case dot:
                        return K;
                    default:
                        token.back();
                        return FinalState.integer;
                }
            }
            return FinalState.error; //can't end the code with an operator
        }
    },
    ...
}
```