词法分析程序实验报告

高露

141250037

目录

a) Motivation/Aim

本实验是自己实现的词法分析程序，目的在于通过编码加深对于词法分析程序原理的理解，加深对于各个步骤算法的印象。

b) Content description

程序的输入为一段程序代码，通过本词法分析程序判断出代码中的常数、标识符、保留关键字、运算符以及非运算符（连接符），输出结果为Token序列，并存储于txt文件中。代码也可对一些简单的异常进行报错提示。

c) Ideas/Methods

主要思想是：①自己定义一些正则表达式RE
②将REs转化为NFAs
③将NFAs合并成为一个NFA
④将NFA转化为拥有最少状态的DFA
⑤基于DFA0进行编码

d) Assumptions

假设程序输入为正常的一段java程序代码。
RE的定义如下：
digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

letter -> A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

operator -> + | - | * | / | % | > | < | = | & | | | ~ | >= | <= | == | != | && | || | ++ | — | += | -= | (|) | [|] | . | ”

separator-> , | ; | { | }

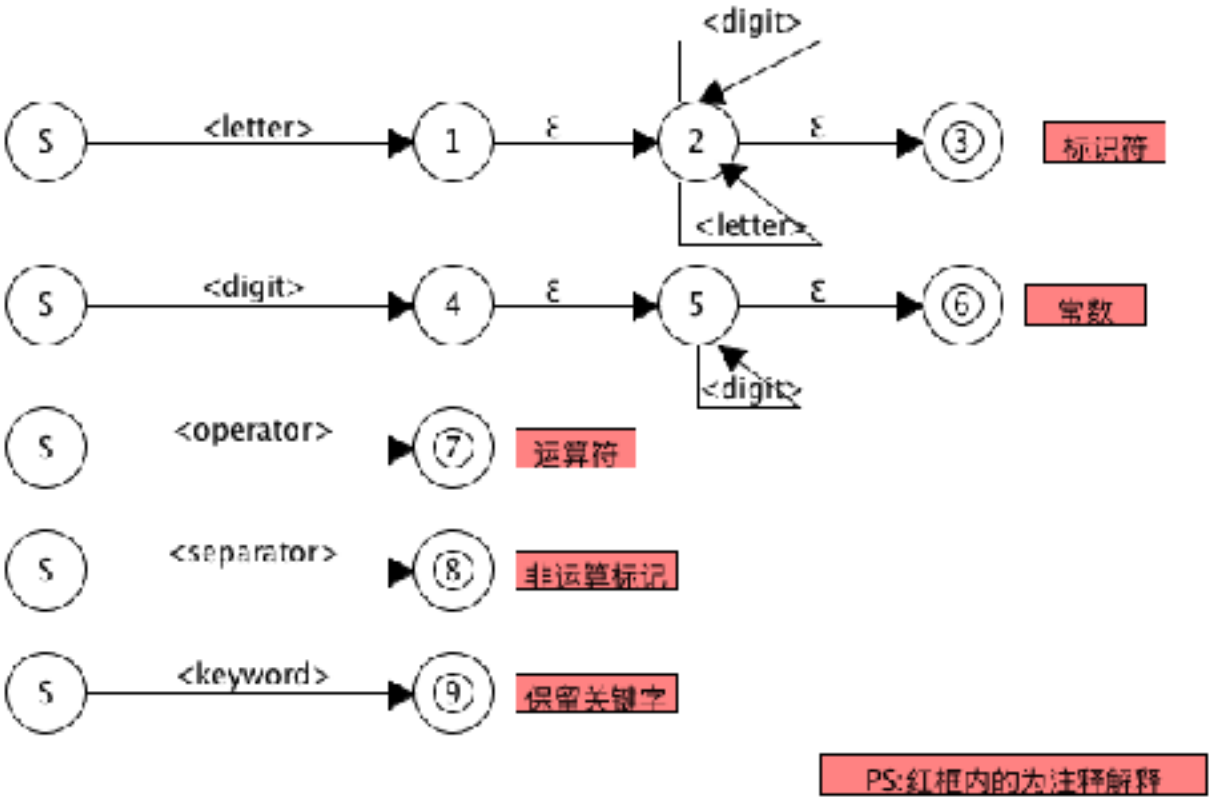
keyword -> abstract | boolean | break | byte | case | catch | char | class | const | continue | default | do | double | else | enum | extends | false | final | finally | float | for | if | implements | import | int | interface | long | new | null | package | private | protected | public | return | short | static | super | switch | this | throw | throws | try | true | void | while

ID -> <letter>(<letter> | <digit>)*

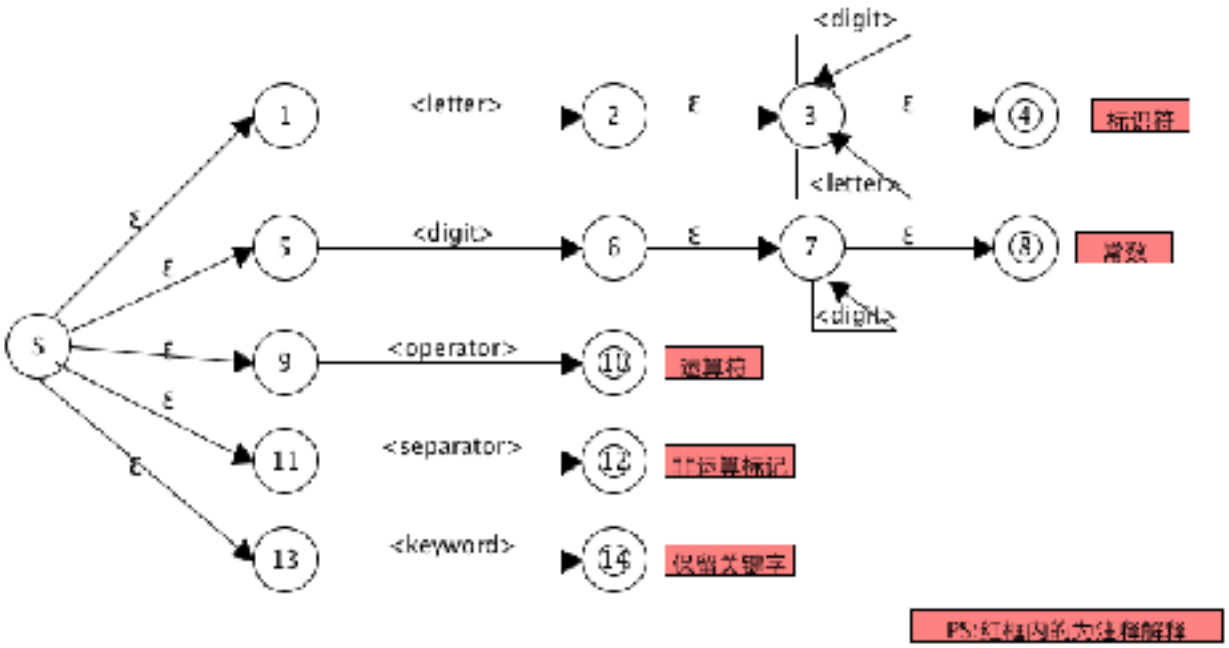
NUMBER -> <digit> (<digit>)*

e) Related FA descriptions

①RE的定义见上
②REs ->NFAs

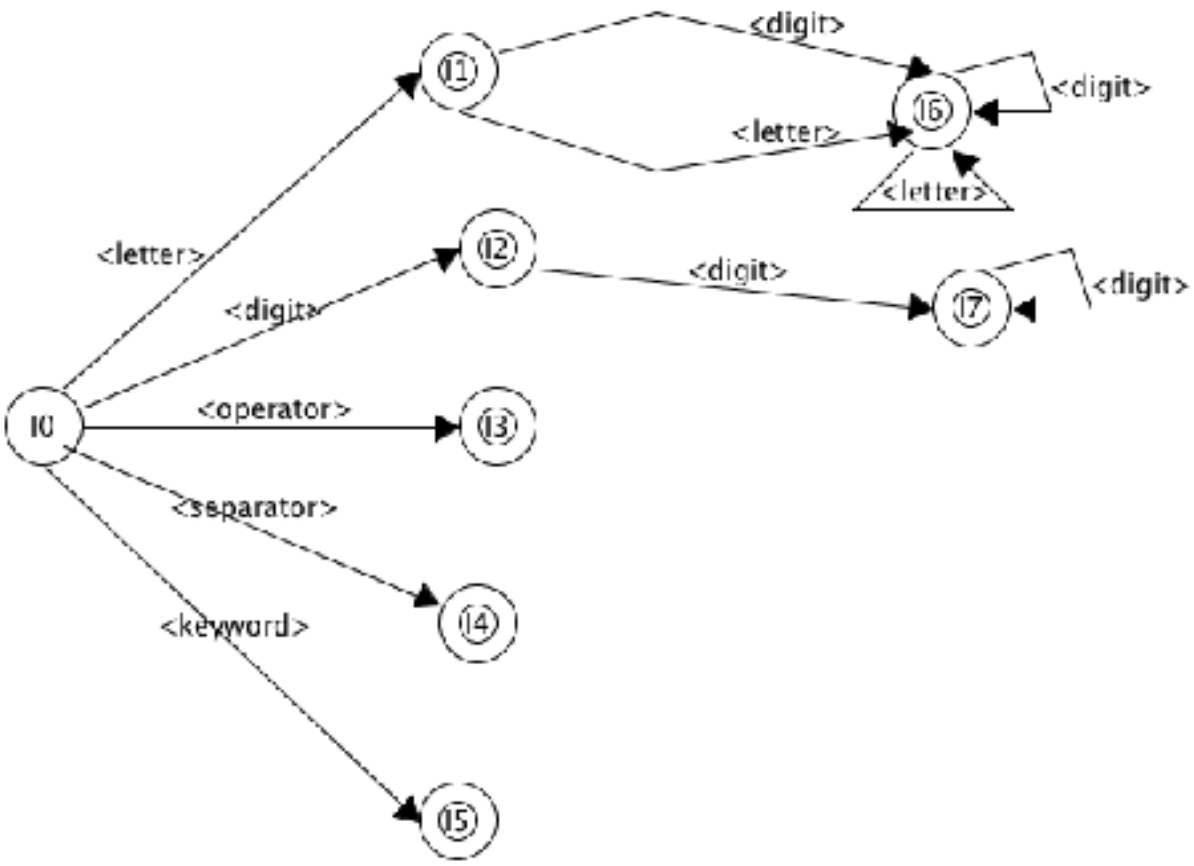


③NFAs -> NFA

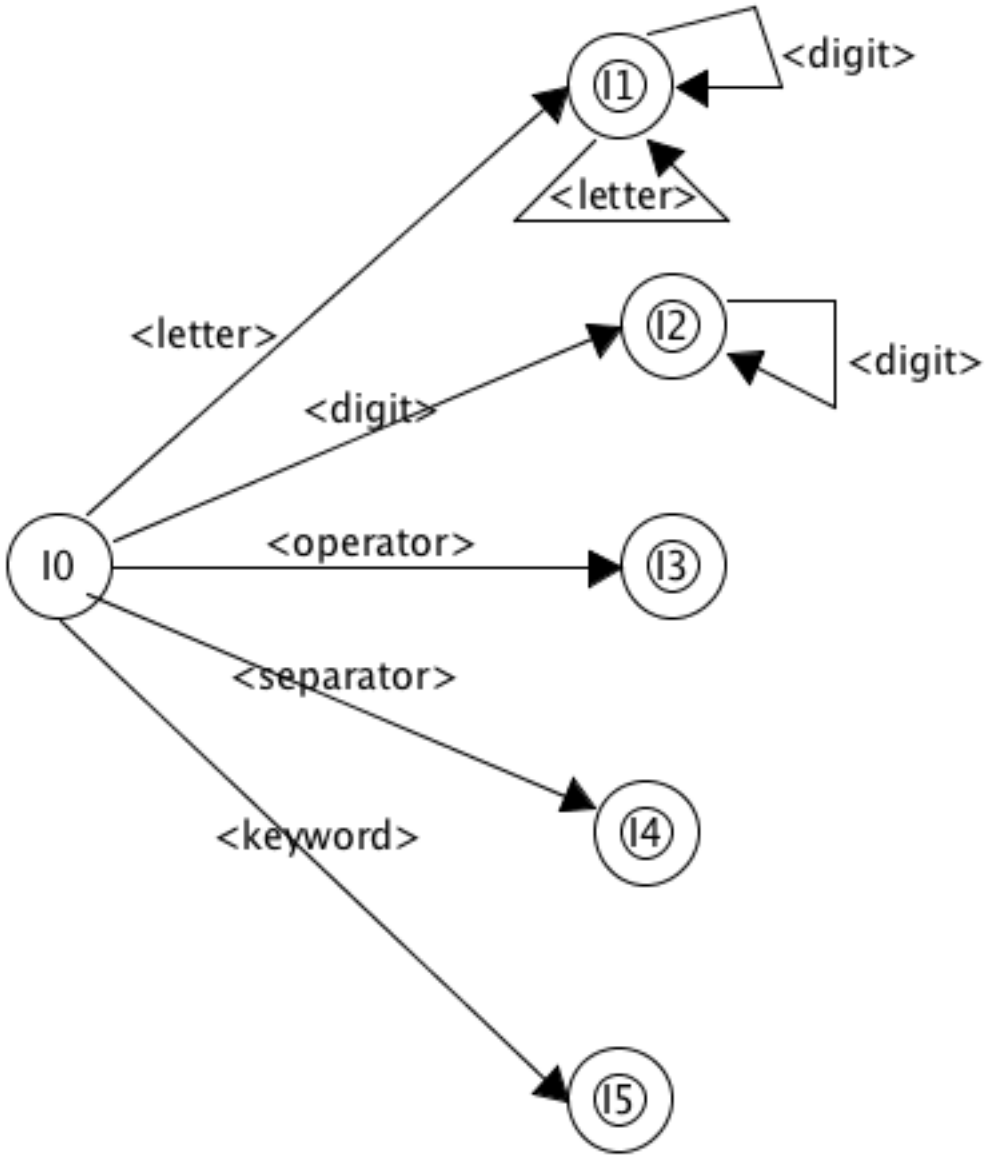


④NFA -> DFA

I	<letter>	<digit>	<operator>	<separator>	<keyword>
I0={S, 1,5,9,11,13}	I1={2,3,4}	I2={6,7,8}	I3={10}	I4={12}	I5={14}
I1={2,3,4}	I6={3,4}	I6={3,4}			
I2={6,7,8}		I7={7,8}			
I3={10}					
I4={12}					
I5={14}					
I6={3,4}	I6={3,4}	I6={3,4}			
I7={7,8}		I7={7,8}			


<digit>
<digit>
<letter>
<letter>
<letter>
<digit>
<digit>
<digit>
<operator>
<separator>
<keyword>
I0
I1
I2
I3
I4
I5
I6
I7

⑤DFA -> DFA0



f) Description of important Data Structures

代码中的Type类定义了各种类别的具体值内容

```
public static final String[] keyword = {"abstract","boolean","break","byte","case","catch","char","class",
                                        "const","continue","default","do","double","else","enum","extends",
                                        "false","final","finally","float","for","if","implements","import",
                                        "int","interface","long","new","null","package","private","protected",
                                        "public","return","short","static","super","switch","this","throw",
                                        "throws","try","true","void","while"};

public static final char[] separator = {',',';','{','}'};

public static final String[] operator = {"+","-","<","/","%",">","=","*","&","|","^","<<",">>",">>>","!=",
                                         "&&","||","++","--","+=","-=","*=","(",")","[","]",".","\""};
```

g) Description of core Algorithms

FileUtil类：read()方法是用来读取input文件，将文件读至一个缓冲区内(StringBuffer)；
Write()方法是用来将结果写入output文件。

Lexer类：analyse()方法是具体的词法分析部分，根据DFA的状态图，定义出5个不同的状态，然后根据每次取到的char的值判断下一步到达什么状态。当达到终态时，将结果写入输出文件，然后将状态重新置回初始的0状态，然后对下个词再进行词法分析。

如以DFA中的State1为例，根据下一个字符的情况进行状态跳转，如果不是字母或数字表明是下一个词，重新置为开始状态。

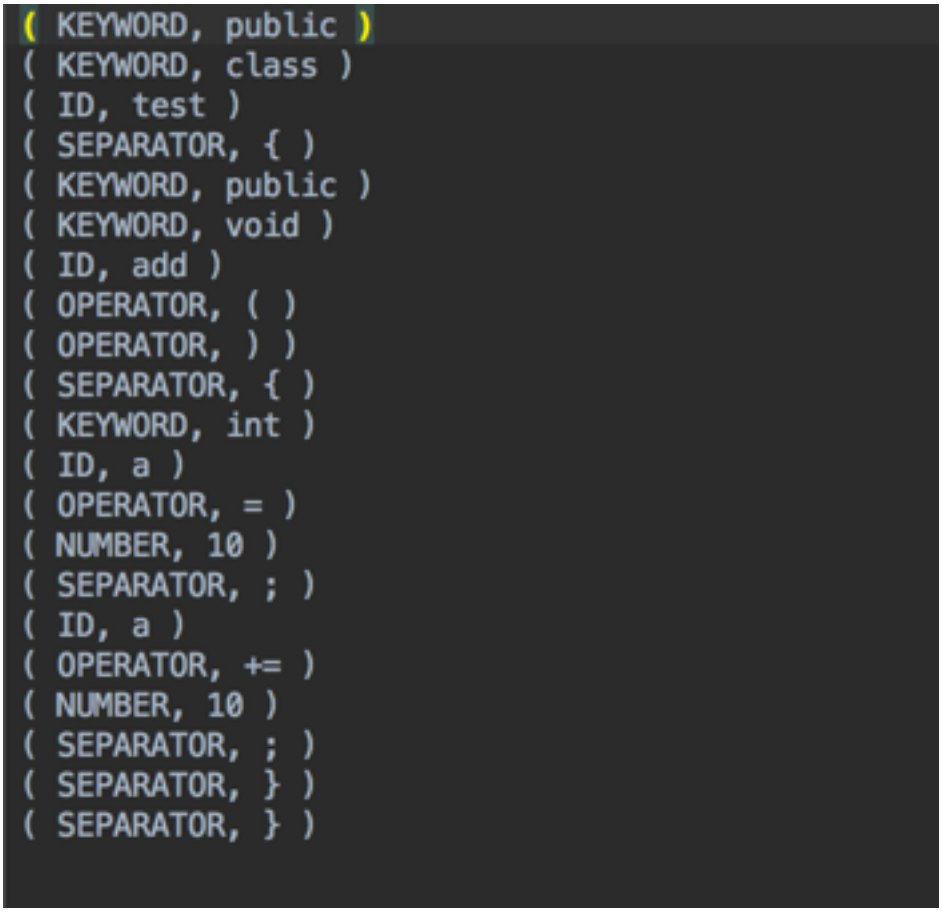
```
case STATE1:
    if(Type.isDigit(ch) || Type.isLetter(ch)) {
        word += ch;
    } else {
        state = State.STATE0;
        if(Type.isKeyword(word.trim())) {
            //System.out.println("(KEYWORD, "+word.trim()+")");
            fileUtil.write("KEYWORD",word);
        } else {
            //System.out.println("(ID, "+word.trim()+")");
            fileUtil.write("ID",word);
        }
        word = "";
        i--;
    }
    break;
```

h) Use cases on running

输入的例子在input.txt中，

```
public class test {
    public void add() {
        int a = 10;
        a += 10;
    }
}
```

输出的情况存储在输出文件output.txt中



i) Problems occurred and related solutions

①多个字符组成的运算符的情况（例如+=）：最终的解决方法是如果是运算符不立即跳转到另一个状态，而是继续读取下一个字符，如果组合起来还是运算符就继续，否则将原来的运算符存储
②按照我的代码写法最后一个字符的情况会无法进入循环使得少一个字符的情况，最终的解决方法比较蠢，见代码。。。
③异常处理情况考虑的比较少，我能想到的只有数字后面不能直接出现字母TAT。。。而且注释的情况暂时没有考虑

j) Your feelings and comments

本来不太明白作业的意思，后来经过查阅一些博客等资料，然后亲自编写程序，感觉对于词法分析程序的理解加深了，受益匪浅！