Compilers Principles Lab

Lab 1 Lex

71118415 叶宏庭

1. Motivation

该实验的目的是为了自行编写一个词法分析器 Lex，可以针对输入的字符流进行词法分析，返回结果的 Token 序列。

2. Content description

1) Input
   Stream of characters
   REs(The number of REs is decided by yourself)
2) Output
   Sequence of tokens
3) Classes of words are defined by yourself
4) Error handling may be included

3. Ideas/Methods

1) 定义正则表达式 RE
2) 自定义 Res -> NFAs -> NFA -> DFA -> DFA°
3) 基于 DFA°进行编码

4. Assumptions

1) 本实验完成的为 java 的词法分析器，因此输入必须为 xxx.java 文件。
2) 定义 RE 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-> , | ; | { | } | \ | \\ | ’ | ” | \t | \n

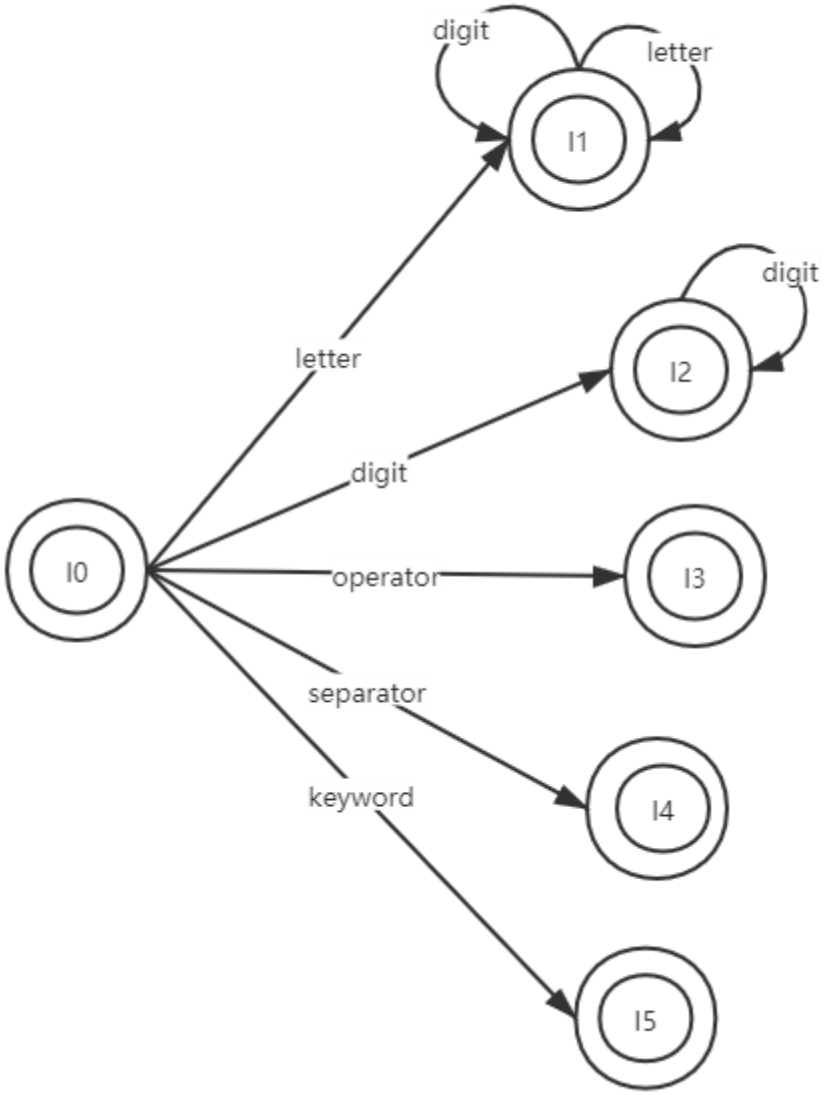
keyword -> abstract | boolean | break | byte | case | catch | char | class | const | continue | default | do | double | else | enum | extends | false | final | finally | float | for | if | implements | import | int | interface | long | new | null | package | private | protected | public | return | short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

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

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

5. Related FA descriptions

最终 DFA°如下图所示：I0 为初始状态



6. Description of important Data Structures

1) **Enum_Type:** 定义了统一的枚举类型, 包括 digit、letter、operator、separator、keyword。
2) **Type：** 存储上诉五种类型的具体 list，包括判断 ch 所属类型的函数 TypeJudge()。
3) **State：** 定义了当前状态的枚举类型，根据 DFA°，共包含六个状态。
4) **tansM 字典**：定义状态表（Soft coding）。

7. Description of core Algorithms

LexScan()方法为词法分析主程序，下图中已经针对程序中的每一步做出解释。

```
159  def LexScan(in_file_path, out_file_path):
160      ch_stream = FileRead(in_file_path)    读取源文件
161      TokenList = []
162      nowState = State.State0      定义初始状态
163      word = ''
164      for ch in ch_stream:      根据字符流进行循环
165          if word == "" and ch==" ":  跳过多余空格
166              continue
167
168          if(not getNextState(nowState, ch)): 根据ch无法到达下一状态（ch为空，或无法进行状态转换，应当结束当前词）
169              if word in Type.keyword:
170                  TokenList.append("<KEYWORD, {0}>".format(word))  若当前词为关键字，则识别为关键字
171              else:
172                  TokenList.append("<{0}, {1}>".format(getStrOFstate(nowState), word)) 当前词不为关键字，识别为对应RE类型
173
174              nowState = State.State0   开始下一轮词识别
175              word = ''
176              if not ch==" ":
177                  word = ch                  回退已经读入的应当属于下一词的ch
178                  nowState = getNextState(nowState, ch)
179          else:
180              nowState = getNextState(nowState, ch)   状态转换
181              word += ch
182
183      TokenList.append("<{0}, {1}>".format(getStrOFstate(nowState), word))    TokenList中加入最后一个词
184
185      FileWrite(out_file_path, TokenList)    输出TokenList
```

输入输出模块：

```
'''
    IO function:
        -- FileRead(): 读取源码
        -- FileWrite(): 输出Token序列
'''
def FileRead(input_file_path):
    with open(input_file_path, 'r', encoding='UTF-8') as f:
        input_str = f.read()
        out_str_1 = input_str.replace("\n", "")
        out_str_2 = out_str_1.replace("\t", "")
        return out_str_2

def FileWrite(output_file_path, TokenList):
    with open(output_file_path, 'w') as f:
        for i in TokenList:
            f.write(i + "\n")
```

状态转换表（Soft coding 关键）：

```
transM = {
    State.State0 : {
        EnumType.letter : State.State1,
        EnumType.digit : State.State2,
        EnumType.operator : State.State3,
        EnumType.separator : State.State4,
        EnumType.keyword : State.State5
    },
    State.State1 : {
        EnumType.digit : State.State1,
        EnumType.letter : State.State1
    },
    State.State2 : {
        EnumType.digit : State.State2
    },
    State.State3 : {
        EnumType.operator : State.State3
    },
    State.State4 : {

    },
    State.State5 : {

    }
}
```

8. Use cases on running

s.java 作为源文件输入，进行词法分析，下图为 s.java 的内容。

```
public class A{
    public void main(){
        char a1 = 10;
        int a2 = 5;
        int a3 = a1 + a2;
        a3 += 5;
    }
}
```

下面给出词法分析结果 out.txt:

<KEYWORD, public>
<KEYWORD, class>
<ID, A>
<SEPARATOR, {>
<KEYWORD, public>
<KEYWORD, void>
<ID, main>
<OPERATOR, ()>
<SEPARATOR, {>
<KEYWORD, char>
<ID, a1>
<OPERATOR, =>

<DIGIT, 10>
<SEPARATOR, ;>
<KEYWORD, int>
<ID, a2>
<OPERATOR, =>
<DIGIT, 5>
<SEPARATOR, ;>
<KEYWORD, int>
<ID, a3>
<OPERATOR, =>
<ID, a1>
<OPERATOR, +>
<ID, a2>
<SEPARATOR, ;>
<ID, a3>
<OPERATOR, +=>
<DIGIT, 5>
<SEPARATOR, ;>
<SEPARATOR, }>
<SEPARATOR, }>

9. Problems occurred and related solutions

1) 在第一次编码时，忘记做 ch 的回退操作，导致部分字串的词法分析丢失。后来补上相应代码后，成功完成词法分析
2) 在定义数据结构时，没有很好组织结构关系，可以进行更好的结构优化。

10. Your feelings and comments

通过本次实验，更加深入理解了 Lex 词法分析器的工作原理，同时完成了自己的一个简易的词法分析器，更好的掌握了相关的理论知识，在未来的学习和工作中都会有很大的帮助，也希望我能够有时间继续去完善这个词法分析器