编译原理词法分析实验报告

石若非 141250108

A． Motivation/Aim

本实验的目的是通过 Java 程序实现一个精简的词法分析器，可以对一个自定义的精简编程语言集进行基本的词法分析。该自定义的编程语言包括基本的保留字、运算符、操作符、关键字和数字等。词法分析的原理是由词素的正则表达式定义转化为 DFA 形式的有限状态机，再由程序进行实现。本次实验是对第 3 章内容的一次实践。

B． Content description

本实验报告主要描述了实现的语法分析器的基本功能和原理，包括正则表达式和有限状态机到实际程序的转化和一些算法的细节。

C． Ideas/Methods

首先将词素进行基本的分类：

保留字：一些基本的语句所必须的保留字，如 if,else,for,int,double 等

标识符：变量名、函数名等标识符，使用 java 中的要求，即字母开头，由字母、数字或下划线组成

数字：整数、浮点数、可带负号

运算符：+、-、*、/

比较运算符：>、<、=、>=、<=、==

其他符号（词素分隔）：;、()、{}

字符串：这里指双引号内的内容，如”HelloWorld!”

程序的基本方法是将源代码转化为字符流，逐一读取每个字符。由以上词素的基本定义

可以写出他们的正则表达式，经过一系列步骤转化为 DFA 后，便可以在程序中设置代表当前状态的变量，并通过一系列 switch-case 结构来模拟该有限状态机，来分辨每个词素的开始与结束及它们的类型。

一些特殊判断：

保留字：因为假设的保留字形式上均为普通的“单词”，即是符合标识符结构的，所以虽然保留字有自己的正则表示，但在程序实现时不做特殊判断，最后在所有标识符中筛选出与保留字表匹配的部分，以此简化有限状态机的复杂度

带负号数字：因在扫描到“-”时无法直接判断是数字前的负号，所以在所有词素列出后进行回溯整理

注释：本实验假设的输入语言不支持注释，所以在有限状态机和程序实现中并未考虑对注释的识别，可能会直接判断为错误词素

D． Assumptions

编程语言：JAVA

运行环境：jdk1.8 版本

自定义词素列表：

词素类型	**示例/枚举**
保留字	if；else；while；for；do；break；continue；int；void；double；char；return；main
标识符	abc；a12345；a_bc12（同 java 中的定义）
数字	21；2.33；-3.1415（正负整数、浮点数）
运算符	+；-；*；/
比较运算符	>；<；=；>=；<=；==

其他词素分隔符	(;) ; { ; } ; ;
字符串	“Hello World!”
错误词素	1.2.3.4；1abc；#$&%&^%

E． Related FA descriptions

- 一些单个字符的正则定义：

letter -> a-z | A-Z

digit -> 0-9

operator -> + | - | * | /

other symbol -> ; | (|) | { | }

- 整体词素的正则定义：

id -> letter (letter | digit | _)*

number -> (digit)? (.)? (digit)*

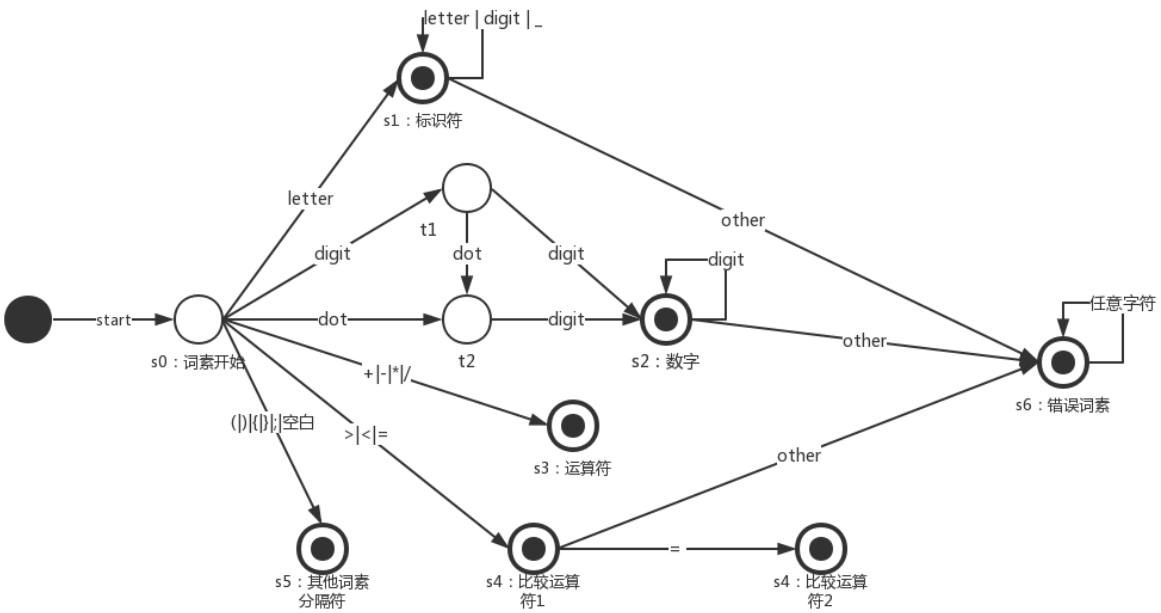
operator -> + | - | * | /

comparison -> (> | < | =) (=)?

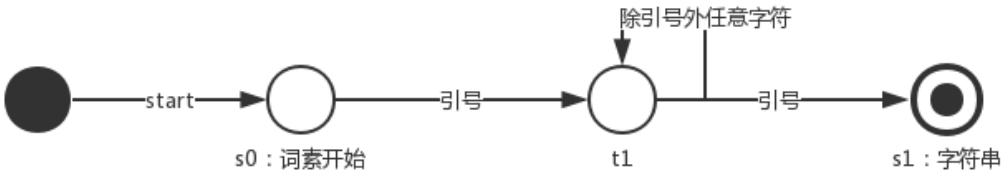
string -> “ (.*) “

other symbol -> ; | (|) | { | }

- 总体 DFA 状态转换图：



其中中间状态 t1、t2 若输入除 digit 外任意字符也会直接到达 s6 错误词素状态



上图为字符串状态转换图，为防止整体状态图过于复杂而单独列出

本实验由以上正则定义与 DFA 有限状态机为算法基础进行编程

F． Description of important Data Structures

- ArrayList<String> id_table

  定义的标识符表，程序最后会将词素中相同的标识符用指针指向标识符表的同一位置

- String reserved_word_table

  定义的保留字表，在程序初始化时便存有所有保留字的 string 形式

- ArrayList<Tokens> tokens

  词素表，Token 类有两个属性，分别为 String type 表示词素类型，String attr 表

示词素属性。词素表存储了分析出的全部词素。

G． Description of core Algorithms

- characterType(char c)函数用于判断当前字符的类型，包括字母、数字、小数点、下划线、运算符、比较运算符、引号、括号、大括号、分号、空白、制表或换行。
- lexHandle()函数则是由有限状态机转化得来的核心程序，通过 switch-case 语句实现了对 DFA 的代码实现。其中临时变量 String str 用于存储当前进行扫描与判断的词素，便于在该词素结束时将其完整地存入词素表中；而变量 int status 则通过不同的整数值代表当前词素不同的状态，在这里的定义为：0 为词素开始;1 为 id 或保留字;2 位 number;3 为 operator;4 为 comparison;5 为引号内容；6 为错误词素;7 为各种分隔符。在个字符进行的 case 判断中，通过对 status 的改变和 str 的重新赋值来实现对有限状态机的模拟，以此实现词素分析的功能。

H． Use cases on running

输入文件为根目录下的 language.txt，内容如下：

```
int main(){
        int a;
        int b = 20;
        double c = -3.14;
        int 123asc;
        double d = 1.1.1;
        if(b>=0)
                a = a + -3.14;
        for(int i=0;i<b;i=i+1){
                c = c/i*a;
        }
        printf("HelloWorld!");
}
```

其中包括了一个简单程序的基本定义、赋值、判断、循环、算术运算、调用函数等操作

结果文件为根目录下的 result.txt，内容如下：

```
< reserved word , int >
< reserved word , main >
< other symbol , ( >
< other symbol , ) >
< other symbol , { >
< reserved word , int >
< id , a id_table[0] >
< other symbol , ; >
< reserved word , int >
< id , b id_table[1] >
< operator , = >
< number , 20 >
< other symbol , ; >
< reserved word , double >
< id , c id_table[2] >
< operator , = >
< number , -3.14 >
< other symbol , ; >
< reserved word , int >
< error , 123asc >
< other symbol , ; >
< reserved word , double >
< id , d id_table[3] >
< operator , = >
< error , 1.1.1 >
< other symbol , ; >
< reserved word , if >
< other symbol , ( >
< id , b id_table[1] >
< comparison , >= >
< number , 0 >
< other symbol , ) >
< id , a id_table[0] >
< operator , = >
< id , a id_table[0] >
< operator , + >
< number , -3.14 >
< other symbol , ; >
< reserved word , for >
< other symbol , ( >
< reserved word , int >
< id , i id_table[4] >
< operator , = >
< number , 0 >
< other symbol , ; >
< id , i id_table[4] >
< comparison , < >
< id , b id_table[1] >
< other symbol , ; >
< id , i id_table[4] >
< operator , = >
< id , i id_table[4] >
< operator , + >
< number , 1 >
< other symbol , ) >
< other symbol , { >
< id , c id_table[2] >
< operator , = >
< id , c id_table[2] >
< operator , / >
< id , i id_table[4] >
< operator , * >
< id , a id_table[0] >
< other symbol , ; >
< other symbol , } >
< id , printf id_table[5] >
< other symbol , ( >
< string , "HelloWorld!" >
< other symbol , ) >
< other symbol , ; >
< other symbol , } >
```

其中包含对输入代码中词素的完整分析，包括标识符（标识符名称以及指向标识符表的指针）、数字（在用例中包括了整数、浮点数和负数）、运算符、比较运算符、保留字、字符串和其它分隔符

Ⅰ. Problems occurred and related solutions

- 保留字处理

  在上文中提到，程序初始化时便有一个存储了所有保留字的数组。而为了降低算法复杂性，实现有限状态机时并未特别考虑保留字的存在，而是利用了保留字的形式符合标识符条件的前提，在对词素进行完整提取后，特别对标识符进行保留字匹配判断，因此得以对算法进行简化

- 字符串处理

  本实验对字符串的定义为双引号中的内容，因为考虑到“成对的引号未出现变换行”的特殊情况，在算法中并未将字符串状态特别列为一个持续的状态，而是在出现第一个引号时，向之后的字符流进行“预测”，若出现成对引号，则将引号内的部分转化为字符串词素进行存储，并继续向下分析；若未出现成对引号，则将指针“回溯”，单独存储引号，同时对接下来的字符重新进行词素分析。

  这个算法可以在有限状态机处于其他状态时不必考虑自己在不在引号中，简化了大量复杂且冗余的代码。

- 负数判断

  在出现字符“-”时，应考虑到，这可能不是一个减号，而是一个在某个数字前，代表取负操作的负号。而这个判断在顺序读取字符流中是难以完成的，即使进行“预测”，也要对之后词素是否能以数字状态结束进行判断，是个复杂度较高的运算。因此在我的实际实现中，是在每个数字词素存储后，额外判断之前的一个词素是否为“-”，若是，则整合至同一个词素。

  该算法的优势依然是能进行有一定程度的简化。然而若在全部词素存储后再次遍历会浪费开销（寻址和可能的数组移动），但若在每个 number 属性的词素刚存储时便进行判断便可避免该问题

J．Your feelings and comments

该程序的算法从整体上来看有一定复杂度，我以前写过的一个计算文件有效代码行数（去除注释、空行、无分号换行等）的题目便与本次实验有异曲同工之处。而在本次实验的实现流程中，通过 RE->NFA->DFA->program 的过程，则将一个整体复杂的有限状态机进行了细化与拆解，这使得虽然之前得步骤繁琐，但在实际编码中，效率与正确

率都比以往“硬写”的方式要有更多优势。

通过本次实验，我对数学模型对算法实现的帮助与铺垫作用有了新的认识。