Report

刘瀚文 131250205

目录

[Report 1](#_Toc435721812)

[1 Motivation and Aim 1](#_Toc435721813)

[2 Content Description 1](#_Toc435721814)

[3 Ideas and Methods 1](#_Toc435721815)

[4 Assumptions 1](#_Toc435721816)

[5 Related FA Descriptions 2](#_Toc435721817)

[6 Description of Important Data Structure 2](#_Toc435721818)

[7 Description of Core Algorithms 2](#_Toc435721819)

[8 Running Method and Result 2](#_Toc435721820)

[9 Problems Occurred and Solutions 3](#_Toc435721821)

[10 Feelings and Comments 3](#_Toc435721822)

# Motivation and Aim

This is a simple program to simulate the prominent software Lex.

# Content Description

In the compressed folder, I have 3 files, which are main.cpp, ex.l, and Report.docx.

**main.cpp**: containing whole source code of the program. You can use C++ compiler to compile it.

**ex.l**: containing a sample lexical definition program for this lab.

**Report.docx**: based on the structure specified in Description on Labs.ppt.

# Ideas and Methods

From regular expression to scanner, I use the common method in lecture.

First, semi-regular expressions are standardized into regular expression. In this process, the optimization of “[A-Z]” is specially taken care of, since transforming it directly to “A|B|C|…|Z” can cause a lot of states being generated in the combination algorithm.

Second, the standardized regular expressions are used to construct ε-NFA. The algorithm is the famous Thompson’s Algorithm.

Third, the ε-NFA constructed by step 2 is transformed to DFA, using subsets construction algorithm.

Fourth, the DFA is minimized by Hopcroft’s Algorithm.

Fifth, transition table is constructed based on DFA.

Sixth, code of the expected lexical analyzer is generated based on transition table and output to file.

# Assumptions

There are few error handling techniques. So the most important premise of rendering correct results from the program is that the grammar of sentences in .l file is correct. Also, you should use C++ compiler to compile main.cpp, and the generated analyzer code.

# Related FA Descriptions

The ε-NFA, DFA, and minimal DFA are constructed by the program.

Node \*\*a = new Node\*[nfastates\*(alphabetsize)];

//pay attention to epsilon(we put it in the 0th col)

Node \*\*d = new Node\*[dfastates\*(alphabetsize)];//pay attention to title for each row(we put it in the 0th col)

These 2 variables represent the NFA diagram and DFA diagram respectively. a and d are pointers to arrays of Node’s. Each line of a represents a state of NFA. Position [i][j] may contain a list constructed by Node, indicating that the ith state can transfer to which of the states after reading the jth symbol in alphabet. Similar solution to d.

# Description of Important Data Structure

Besides the most intricate structure in this program mentioned above, other instances of dynamic arrays and linked list are included in this program.

Also, a basic structure representing set is used when operating subset construction algorithm.

# Description of Core Algorithms

Besides the description in Topic 3, algorithm of transforming infix expressions to postfix expressions is implemented in the program to help the construction of NFA.

Also, when computing next states in NFA, formula is used.

# Running Method and Result

Detail methods are described in main.cpp.

I would like to describe how the generated analyzer is performed.

The **input** file is: if else a21 12 ifa la 2 12a 12.1 a2=5.6

The **output** is:

2 if

2 else

1 a21 8503712

3 12

1 ifa 8504384

1 la 8503936

3 2

3 12

1 a 8504104

5 12.1

1 a2 8504440

6 =

5 5.6

id table: a21 ifa la a a2

The first column indicates the type of word. The second column is the actual name of it. The third column has value if the word is id, pointing to the position in id table.

# Problems Occurred and Solutions

The most significant problem is dealing with sentences like [a-zA-Z]. At first, I simply transforming it to a|b|…|z|A|B|…|Z. But it causes dozens of states being generated by Thompson’s algorithm; and the program has to run for minutes, taking up about 50MB in memory, even though the lexical definition is quite simple. The solution to this problem is that we take special handle of this kind of sentences, which only requires 2 states and n edges to represent a sentence.

Also, some small but annoying bugs like memory violation are solved quite efficiently.

# Feelings and Comments

I learned a lot from this experiment and this course. I am quite interested in computer science. So it is very exciting to get to know something about Language and Automata Theory. Moreover, I learned how to write a program based on proved theories to ensure the correctness of a software.

Thanks!