Project One

Collin Drake

Collin.Drake1@Marist.edu

February 12, 2024

Lab 1

Crafting a Compiler Exercises:

Exercise: 1.11 (MOSS)

The Measure of Software Similarity (MOSS) [SWA03] tool can detect the similarity of programs written in a variety of modern programming languages. Its main application has been in detecting similarity of programs submitted in computer science classes, where such similarity may indicate plagiarism (students, beware!). In theory, detecting equivalence of two programs is undecidable, but MOSS does a very good job of finding similarity in spite of that limitation. Investigate the techniques MOSS uses to find similarities. How does MOSS differ from other approaches for detecting possible plagiarism?

The Measure of Software Similarity tool, commonly referred to as MOSS runs through several steps to determine the possibility of a submission being plagiarism. It starts by removing irrelevant elements from code such as whitespace and identifiers. Secondly, it generates the fingerprints, or unique identifiers of a submission by hashing the input. These fingerprints are then compared to other submissions, counting the frequency of matches to determine the likelihood of plagiarism. This differs from other approaches because it uses tools like hashing and it throws away identifiers and whitespace.

Exercise: 3.1

```
1.  Assume the following text is presented to a C scanner:

main(){
const float payment = 384.00; float bal;
int month = 0;
bal=15000;
while (bal>0){
printf("Month: %2d Balance: %10.2f\n", month, bal); bal=bal-payment+0.015*bal;
month=month+1;
} }

What token sequence is produced?
For which tokens must extra information be
returned in addition to the token code?
```

Token Stream: <ID> <LParen> <RParen> <LBrace> <TypeConst> <TypeFloat> <ID> <Assign> <Number> <EOS> <TypeFloat> <ID> <EOS> <TypeInt> <ID> <Assign> <Number> <EOS> <ID> <Assign> <Number> <EOS> <While> <LParen> <ID> <GreaterThan> <Number> <RParen> <LBrace> <ID> <LParen> <Quote> <StringLiteral> <Quote> <Comma> <ID> <Comma> <ID> <RParen> <EOS> <ID> <Assign> <ID> <Subtract> <ID> <Add> <Number> <Multiply> <ID> <EOS> <ID> <Assign> <Assign> <Add> <Number> <EOS> <RBrace> <RBrace>
Extra Information would be returned for any numbers, identifiers, and string literals.

Dragon

Exercise: 1.1.4

A compiler that translates a high-level language into another high-level language is called a source-to-source translator. What advantages are there to using C as a target language for a compiler?

Using C as a target language for a Transpiler comes with many advantages. One of those being that C is very common, which means that C code can be run on many different devices.

Exercise: 1.6.1

For the block-structured C code of Fig. 1.13(a), indicate the values assigned to w, x, y, and z.

```
1     int w, x, y, z
2     int i = 4; int j = 5;
3     {
4         int j = 7;
5         i = 6;
6         w = i + j;
7     }
8     x = i + j;
9     {
10        int i = 8;
11        y = i + j;
12    }
13    z = i + j;
```

w is assigned 13, x is assigned 11, y is assigned 13, and z is assigned 11,

Lab 2

Crafting a Compiler Exercises:

Exercise: 3.3

Write regular expressions that define the strings recognized by the FAs in Figure 3.33 on page 107.

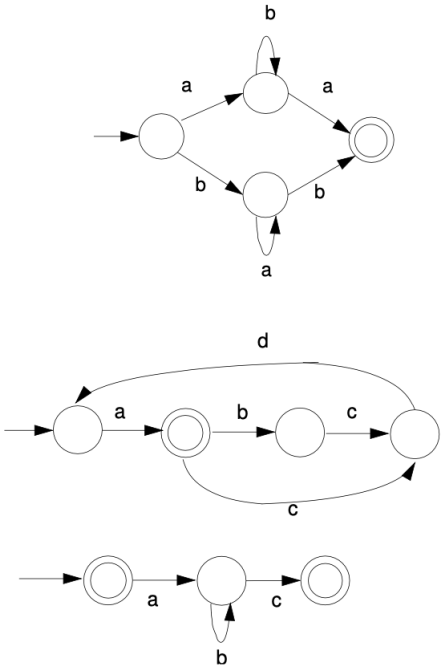


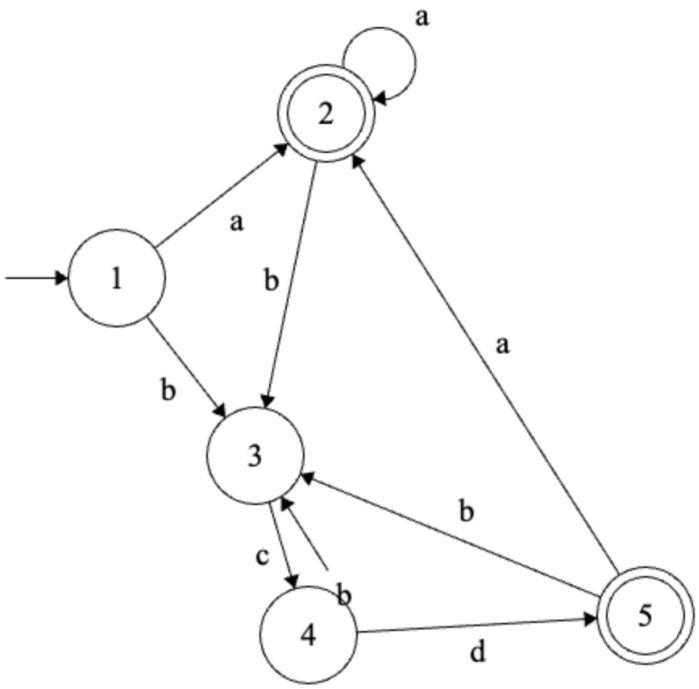
Figure 3.33: FA for Exercise 3.

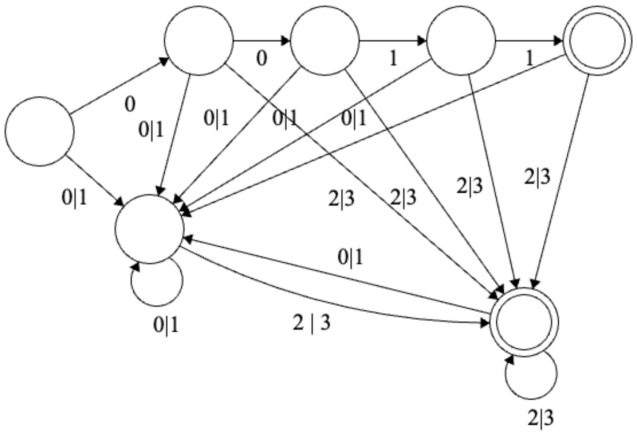
1. (ab*a)|(ba*b)
2. a((bc|c)d)*
3. (ab*c)

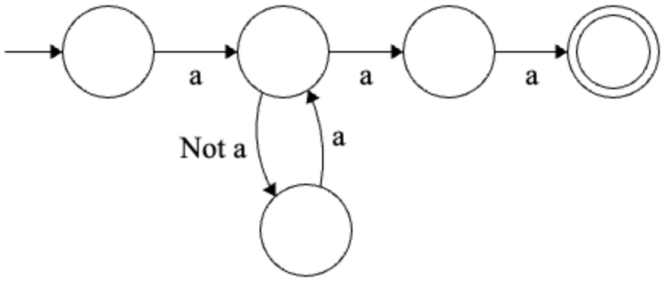
Exercise: 3.4

Write DFAs that recognize the tokens defined by the following regular expressions:

```
1    (a) (a | (bc)* d)+
2    (b) ((0 | 1)* (2 | 3)+) | 0011
3    (c) (a Not(a))* aaa
```


a
2
a
1
b
b
a
3
b
c
b
4
5
d


0
1
1
0
0|1
0|1
0|1
0|1
0|1
2|3
2|3
2|3
2|3
0|1
0|1
2 | 3
2|3



Dragon

Exercise: 3.3.4

Most languages are case sensitive, so keywords can be written only one way, and the regular expressions describing their lexeme is very simple. However, some languages, like SQL, are case insensitive, so a keyword can be written either in lowercase or in uppercase, or in any mixture of cases. Thus, the SQL keyword SELECT can also be written select, Select, or sElEcT, for instance. Show how to write a regular expression for a keyword in a case-insensitive language. Illustrate the idea by writing the expression for "select" in SQL.

Regular Expression: /select/i
The i indicates that the RegEx is case insensitive.

1 Project 1

1.1 Lexical Analysis

A Compiler consists of multiple parts, including a Lexer, also known as a Scanner or Lexical Analyzer. The Lexer's primary function is to turn the source code into an ordered stream of tokens based on the language's grammar rules. Tokens, as described in our slides are "a sequence of characters that we treat as a unit in the grammar of our language." A Token is made up of two components: a type, and the corresponding value represented in the source code. The Lexer's only focus during the compilation process is on the words, symbols, whitespace, and comments used within the source code, not its order or meaning.

To test my lexer I used test cases provided by our professor as well as hall-of-fame projects. You can find the test cases below:

Test Case 1

```
1    {}$
2    {{{{{{}}}}}}$
3    {{{{{{}}} /* comments are ignored */ }}}}$
4    { /* comments are still ignored */ int @}$
5    {
6      int a
7      a = a
8      string b
9      a=b
10   }$
```

Test Case 2

```
1    {}$/*$$$ This should be ignored */$$$"the$tringstillsplits"
```

Test Case 3

```
1    /*Long Test Case - Everything Except Boolean Declaration */
2    {
3    /* Int Declaration */
4    int a
5    int b
6    a = 0
7    b=0
8    /* While Loop */
9    while (a != 3) {
10   print(a)
11   while (b != 3) {
12   print(b)
13   b = 1 + b
14   if (b == 2) {
15   /* Print Statement */
16   print("there␣is␣no␣spoon" /* This will do nothing */ )
17   }
18   }
19   b = 0
20   a = 1+a
21   }
22   }
23   $
```

2 Appendix

Token Class (Constructor)

```
    /*
        Token Constructor
        Creates tokens for the Lexer
    */

    public class Token {
        //each token has a type, lexeme, position, and line
        String tokenType;
        String lexeme;
        String position;
        String line;

        //creating tokens!
        public Token(String tType, String sCode, String line,String position){
            this.tokenType = tType;
            this.lexeme = sCode;
            this.line = line;
            this.position = position;
        }
    }
```

Compiler Class (Entry Point)

```
    /*
      THECompiler entry point
      Builds the parts of a compiler, connects them, and calls them
    */

    public class Compiler {
        public static void main(String[] args)
        {
          //create the parts of a compiler
          Lexer lexer = new Lexer();

          //introductory output
          System.out.println("Welcome to THECompiler by Collin Drake!");

          //check for command line arguments
          if(args.length > 0)
          {
            //file handling and lexer initialization.
            Keep user updated on what happens and keeping naming obvious
            String textFile = args[0];
            System.out.println("Processing file: " + textFile);
            /*
              output to user
              call readfile to read the input text file into a string
              call scanner to the lex the input file
            */
            lexer.readInput(textFile);
            System.out.println("Powering on the LEXER...");
            lexer.scanner();
          }
          else{
            System.out.println("No program found.
            Please provide a command line argument to the compiler.");
          }
        }
    }
```

3 References

3.1 Links

Below are the resources I have used to create simple, readable, and beautiful code.

- Receive input from the command line: geeks4geeks
- Provided access to resources such as slides, textbooks, and code snippets. I utilized some of the code listed under "AlanC, a simple lexer via JavaCC" to help structure my lexer: Labouseur.com
- Basic Regular expressions in Java: geeks4geeks
- Reading an input file into an arraylist: stackoverflow
- Clean up my pushes to git by removing compiled class files: automationpanda
- The "dragon" textbook....: dragon
- Crafting THIS Compiler....: Crafting a Compiler
- RegEx assistance...: tutorialspoint
- RegEx assistance... again...: oracle
- RegEx assistance... the finale...: regex101
- Regex groups: javatpoint
- Regex matches: stackoverflow
- switch statements: w3schools
- Regex finding position: stackoverflow
- Escaping an escape character. Regex: stackoverflow
- Java break and continue: w3schools
- Java exit program: baeldung
- MOSS Understanding: github
- Creating DFAs: madebyevan
- Test Cases: Labouseur.com and Gabriel Arnell