Concordia University

Winter 2019

Assignment 1

Lexical Analyzer Documentation

COMP 442

Written by:

Genevieve Plante-Brisebois

ID:40003112

Presented to:

Dr. Joey Paquet

January 28, 2019

**Lexical Specifications:**

Atomic lexical elements of the language:

id:: = Letter alphanum\*

alphanum:: = letter | digit| \_

integer::= nonzero digit\* |0

float::= integer fraction [e[+|-] integer]

fraction::= .digit\* nonzero | .0

letter::= a.. z |A..Z

digit: 0+1..9

nonzero::=1..9

operator::= == | <> | < |> | <= | >= | + | - | \* | / | = | && | || | !

punctuation :: = ; | , | . | : | :: | ( | ) | { | } | [ | ]

keywords::= if | then | else | for | class | integer | float | read | write |return | main

comments::=/\*\*/ |//

The changes that were made were only to put the operators, punctuation and keywords in their respective token identification and regular expression. The rest of the lexical specifications have been kept as they are.

To my understanding an alphanumeric is a single letter or digit but as there can be any combination of those in the id it creates the alphanumerical characteristic.

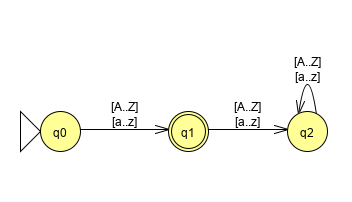
The regular expressions were translated into NFA and then DFA in a manual manner using the knowledge acquired in the COMP 335 class. The conversion to numerical format has been done using the JFLAP software. It is a tool use in COMP 335 which allows to draw the automaton in a clean and clear manner. In the next section, it will be possible to see each individual automata for the regular expression and then the final automata which has all of the lexical specifications in it. The final automata has been built by combining all the automata.

The automata for the keywords is made by using the letter definition. i.e. to got a final state to the if keyword one needs to enter two letters (i then f). This logic has been used for all keywords.

**Finite State Automaton:**

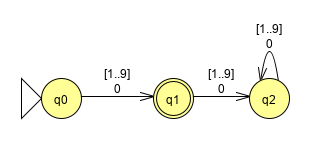
Individual DFA’s for the lexical specifications:

Letter:



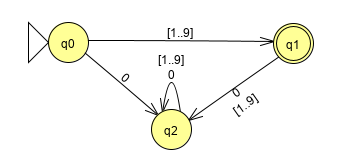
The last state has been added for additional clarity in terms of the fact that only a single char from the presented set is accepted. Else it goes in trap state.

Digit:



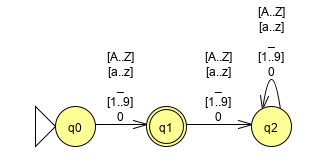
Just like for the letter, the last state has been added for clarity as this is the DFA for a digit and not an integer. If there is more than one digit then it goes in trap state. The terms 0 and [1..9] have been used in order to facilitate the combination of the multiple DFA together as other DFA require both digits and nonzero.

Nonzero:



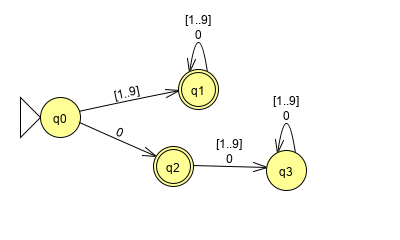
In the case that a zero is used first then all the other attempts made to get a final result will be ending in a trap state. Moreover, as it is a single digit number if there is more than one digit it also ends in a trap state.

Alphanum:

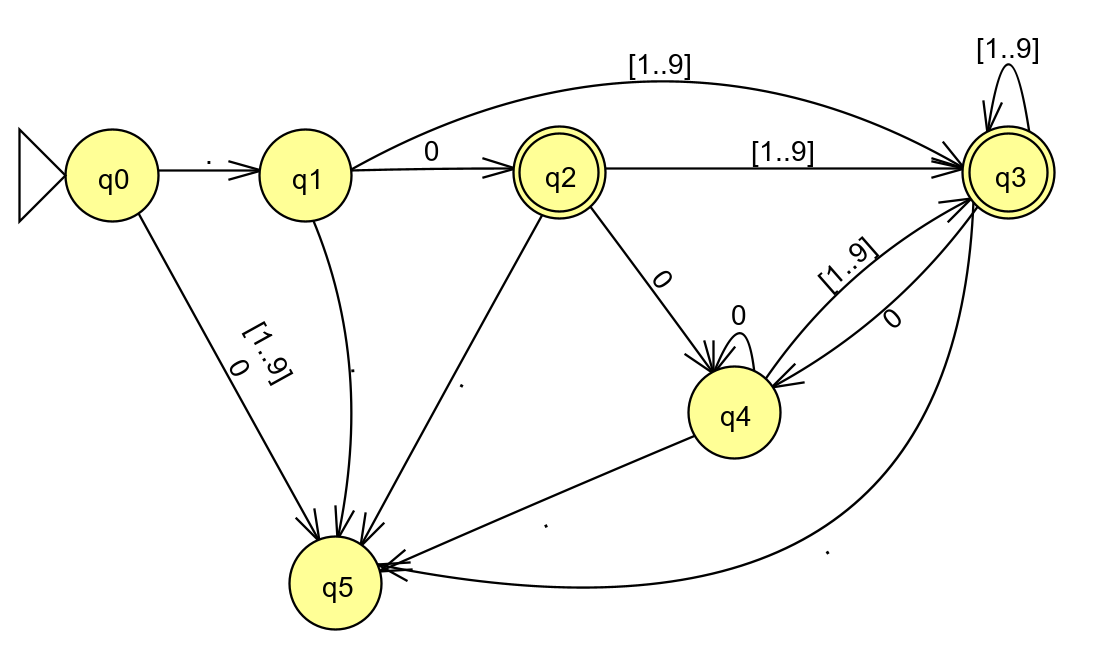


It is my understanding from the regular expression supplied in the lexical specification that the alphanumerical is only one character but it can be a letter, digit or \_ . It is with the \* in the ID that the alphanumerical nature of the string will be made.

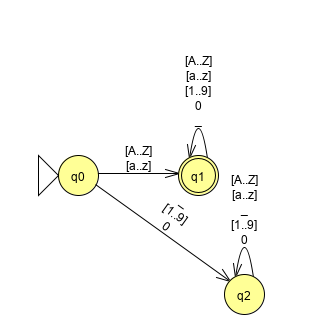
Integer:



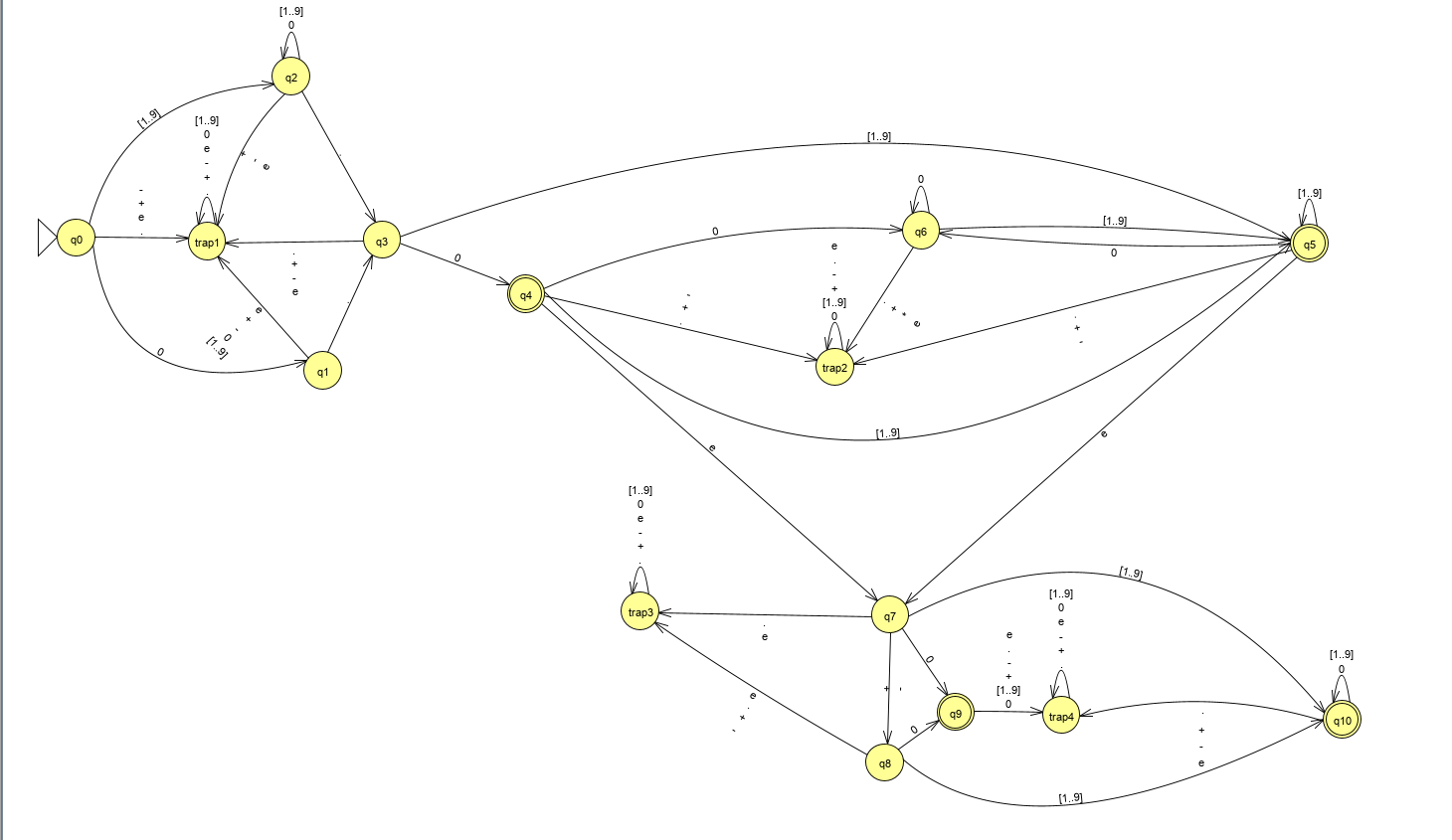
Fraction:



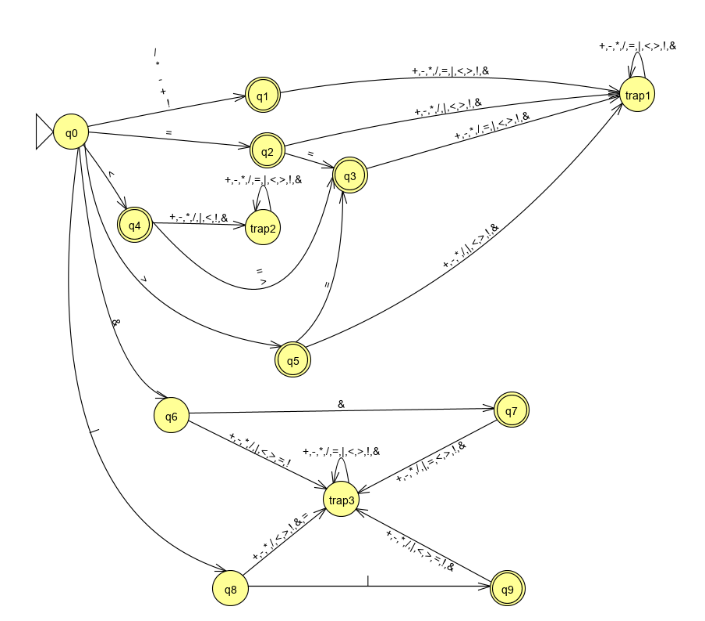
ID:



Float:

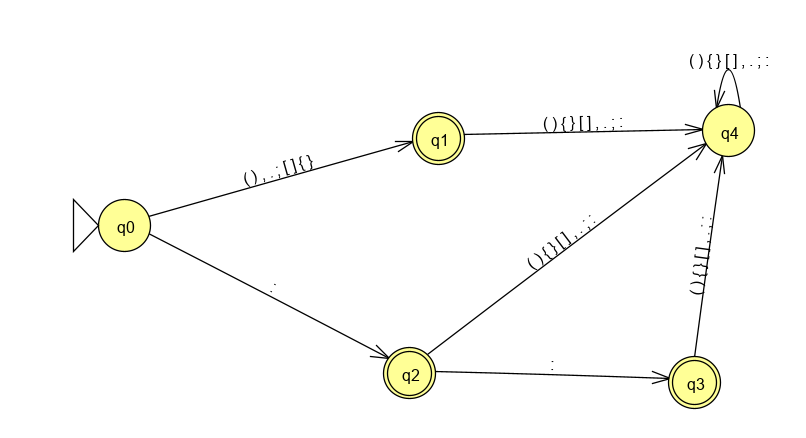


Operator:

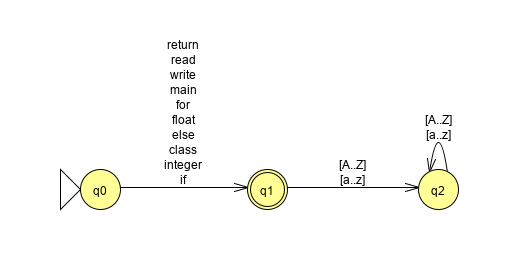


In this graph, the various symbols that can be used are separated by a coma in order to increase readability of the graph. There are ten possible symbols that can be used as operators. They are +,-,\*,/,=,|,<,>,!,& . The various combinations of them are described in the lexical specifications and give this DFA

Punctuation:

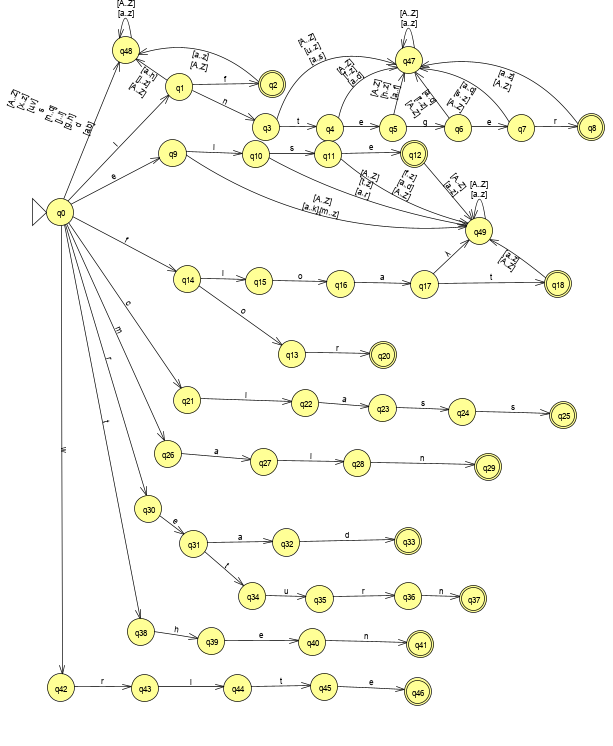


Keywords:

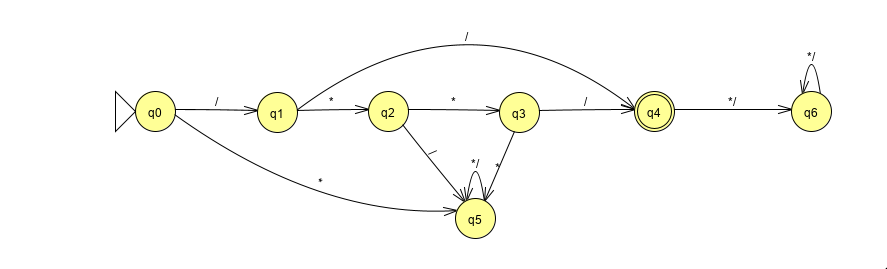


The keywords DFA is quite simple in this format but a more exhaustive DFA could also be done. For the sake of saving time and since it is possible to implement a patterns in Java so to create a bank of keywords that will be recognized when scanning a string this simpler version of this DFA has been drawn. It is also more readable than another model.

Just to show that the other model is rather hard to read here is a partial DFA, so still in NFA state of what it would give to do the exhaustive method:



Comments:



DFA used to make the scanner:

Transition table resulting from the DFA:

**Error Reporting and Recovery:**

In this section it will be possible to find the test cases that are going to be in the driver in order to test the lexer and their expected output. In the case that there is an error there should be an output of either error or invalid token notification.

**Design:**

**Use of Tools:**

Programming language: Java

Programming environment: Eclipse

Version Control: Git

Documentation: Word

Regex, NFA and DFA : JFLAP

SEETING UP THE TOOLS:

JFLAP:

It is only needed to download the program from this link <http://www.jflap.org/> and then, when the program is running, choose the finite automaton format. It will display the tools in order to build one. It is also possible to enter a RegEx and the program will convert it into an NFA. However, conversion to DFA has to be done manually.

Eclipse:

Eclipse is on all the Engineering and Computer Science computer laboratories. I used the default settings of Eclipse.

GitHub:

I created a repository and then imported/exported my program and documentation as needed from this repository.

REASONS OF CHOICE:

The main reasons for the tools used in this assignment are that they are the tools I am the most familiar with. Having entered the class on the last possible date, I had to make sure I understood the past two weeks’ worth of material and completed the assignment in one week while handling other classes and assignment. If there are better tools for the following assignments, I will take the time to look into it now that I have caught up with the material.

Java is the language I am most familiar with and Eclipse is my favorite IDE to code in Java. As for the DFA’s transition model on numerical format, I used JFLAP, a software I had to use by demand of my professor in COMP 335. I find the resulting graphs are very clear and clean. It is also a very versatile and easy to use tool which allows to work quickly. The cons of using JFLAP is that there are a lot of steps which I have to do manually, which increases the chances of errors. However, for efficiency and productivity, I used tools which were more familiar to me.

In the next assignment, it would be good to use AtoCC. It allows to double check the work that is being made. It also auto creates the transition tables and makes the automata related task faster. However in the current assignment I chose to go with a software that I already knew the way it worked even if it meant more manual work. I downloaded the program but I was having a hard time to familiarize myself with it and I mostly wanted to have a functioning lexer for the assignment even if there are more chances of errors in my regular expressions and DFA.

Another tool that could be used in the next assignments is ShareLatex. It can allow for very professional formatting and the AtoCC has functionality with it in order to insert the images of the graphs. However, using ShareLatex is very time consuming compared to Word and as such this will be a deciding factor in which tool I will use for documentation in the next assignments. For the IDE, I will keep using Eclipse unless there is a more recommended IDE by the professor or the TA as we are building on top of each assignment. I will keep a familiar IDE in order to be more efficient when coding.