Documentation Report
Assignment 1: Lexical Analyzer

COMP 442-NN

WINTER 2023

Jeremy Piperni

40177789

1. Lexical Specifications

The specifications stayed the same, but I did perform some slight quality-of-life splits to make my life easier in the future. For example, a digit is just a ‘0’ or a non-zero character. I split the letter lexical element in two. A letter can either be the character e or any letter except ‘e’. I made this split because of the possible ‘e’ that we can have in a float value.

Table 1.1: Letter/Digit/Non-Zero regex

Group	Name	Name in dfa	regex
Digit	Zero	0	0
Digit	Non-Zero	n	[1-9]
Letter	E	e	e
Letter	Letter except e	L	([a-z] | [A-Z]) - e

With the regex specifications shown above, I can now create the regex for alpha-nums and fractions.

Table 1.2: Alphanum/Fraction regex

Name	regex
Alphanum	L | e | 0 | n | _
Fraction	.(0 | n)*n | .0

We can now create the regex for our return types for the language

Table 1.3: ID/Integer/Float regex

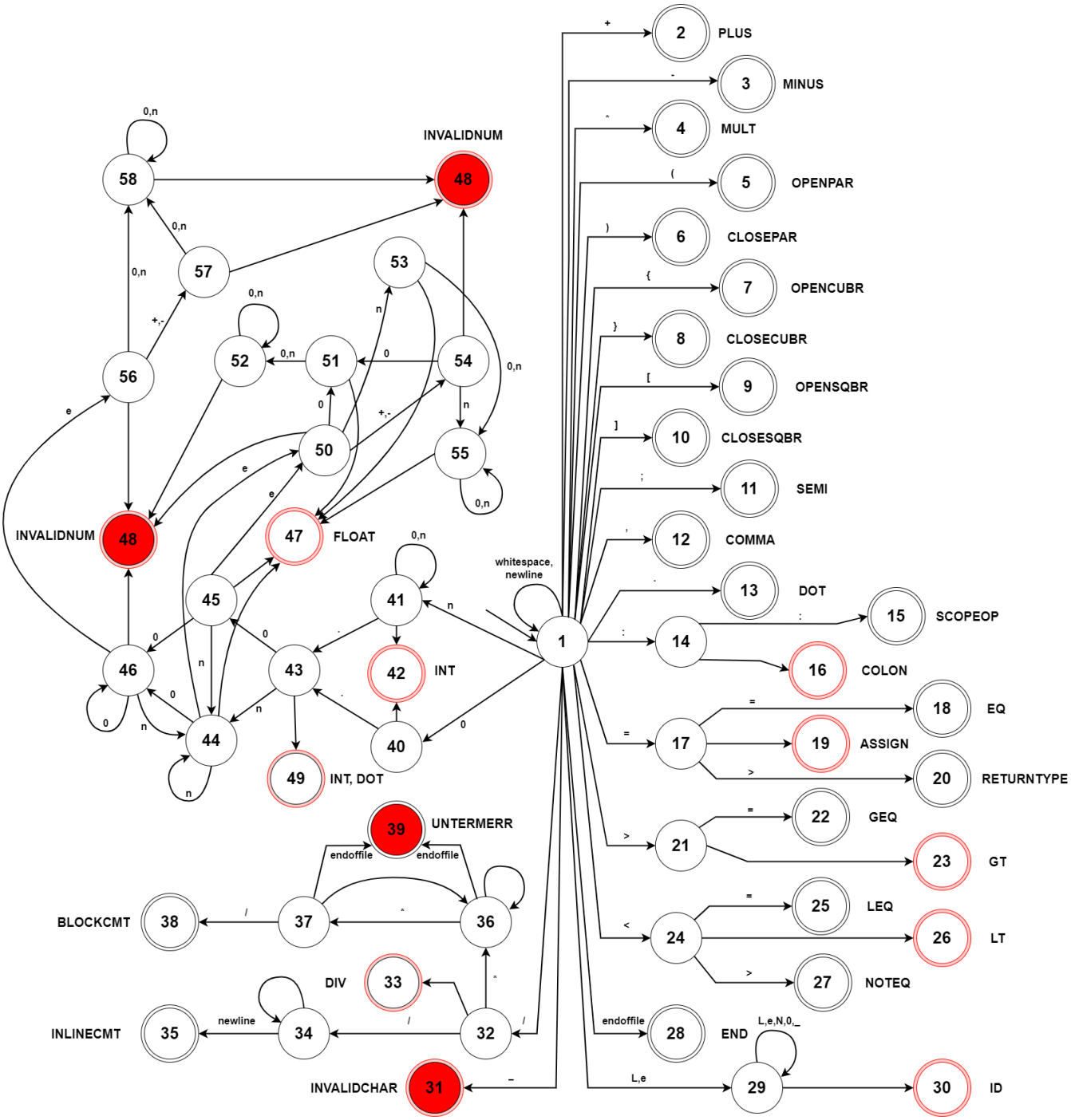
Name	regex
ID	(L | e)(L | e | 0 | n | _)*
Integer	n(0 | n)* | 0
Float	(n(0 | n)* | 0)(.(0 | n)*n | .0)(e(+ | -)?n(0 | n)* | 0)?

Every operator, punctuation, and reserved word is implemented. Comments are also implemented except for nested block comments.

2. Finite State Automata

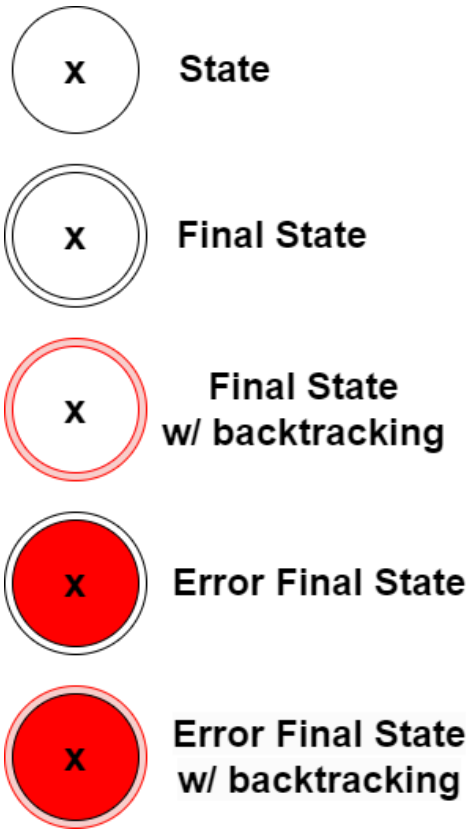
The following DFA (Figure 2.1) was derived from the regex in Section 1. The DFA was derived and produced by hand. The use of a Regex to DFA website was used as will be detailed in Section 4.

Figure 2.1: Deterministic Finite State Automata



The following legend (Figure 2.2) can be used to read the figure. State 48 appears twice in figure 2.1 just for reading simplicity. An empty line represents any other character in the language.

Figure 2.2: DFA Legend



3. Design

I implemented a table-driven scanner with a state transition table. The state transition table was derived directly from the DFA (Figure 2.1). The nextToken() function is called by the lexical driver where it starts reading the first character of the next token. The lexer stores the character read and goes to the next state depending on what it is. The program reads every character until a final state is reached and the program returns the token to the driver.

My program is coded in Java. I chose this language because it is the language I have the most experience with.

4. Use of Tools

Tools used:

1. Regex to DFA converter used for some parts of the DFA construction
   https://cyberzhg.github.io/toolbox/nfa2dfa

Libraries used:

1. JTable: Used to create a table object for the state transition table