Lucas Turpin (40029907) Assignment 1

COMP 442 - Presented to Prof. Joey Paquet on January 27, 2020

Section 1: Lexical Specification

I break down the language specification into 3 parts which can be expressed as the following regular expressions:

Numerical	(0|(1..9)(0..9)*)(.((0..9)*(1..9)|0)(e(-|+)?(0|(1..9)(0..9)*))?)?
Word	(a..z|A..Z)(a..z|A..Z|0..9|_)*
Symbol	(== | = | <> | < | > | <= | >= | . | , | - | * | + | : | :: | ; | { | } | [|] | (|) | / | //$*N? \| /*($|N)**/)

I use the *$* symbol as a wildcard representing any character but the *newline* character, and *N* to represent the *newline* character.

Each of these 3 expressions captures an independent part of the language specification. Identifiers and reserved words are captured as words and the distinction can be established once the full lexeme has been extracted. Integers and floats are also captured uniformly as numericals and the distinction can be established on the full lexeme has been extracted.

Using alternation, the full lexical specification of the language can be formed by the union of the patterns described above:

Token ::= *Numerical* | *Word* | *Symbol*

Section 2: Finite State Automaton

Each of the above regular expression can be converted into an independent DFA:

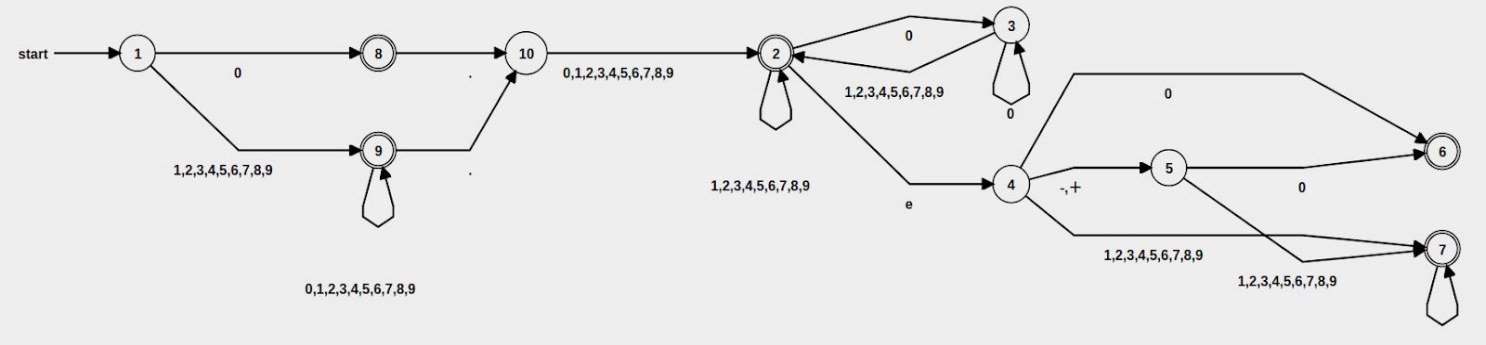

Figure 1: DFA representing *Numerical*

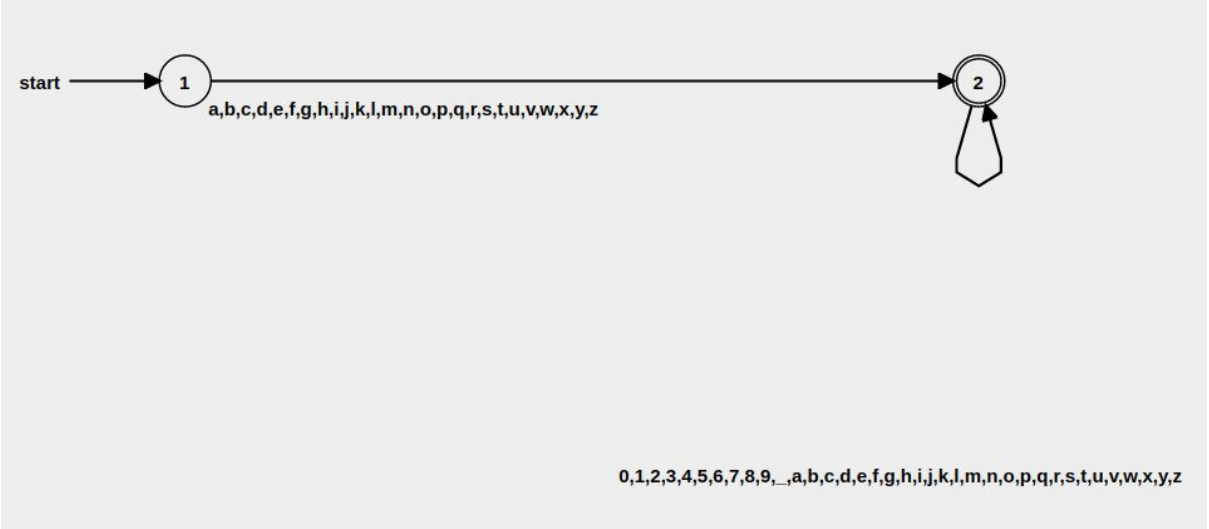

Figure 2: DFA representing *Word*

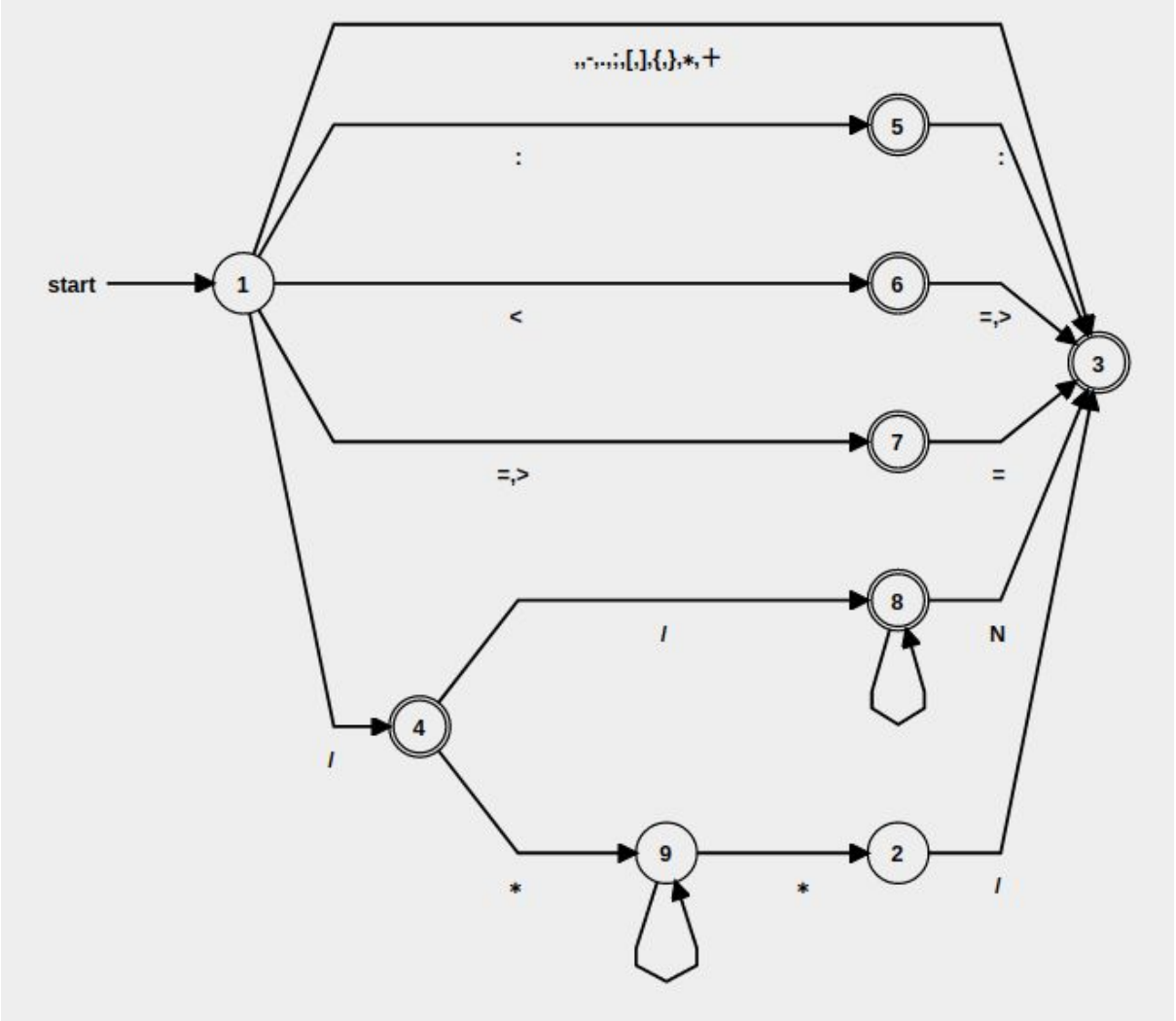

Figure 3: DFA representing *Symbol*

Putting all of the individual DFAs together into a larger one gives the following:

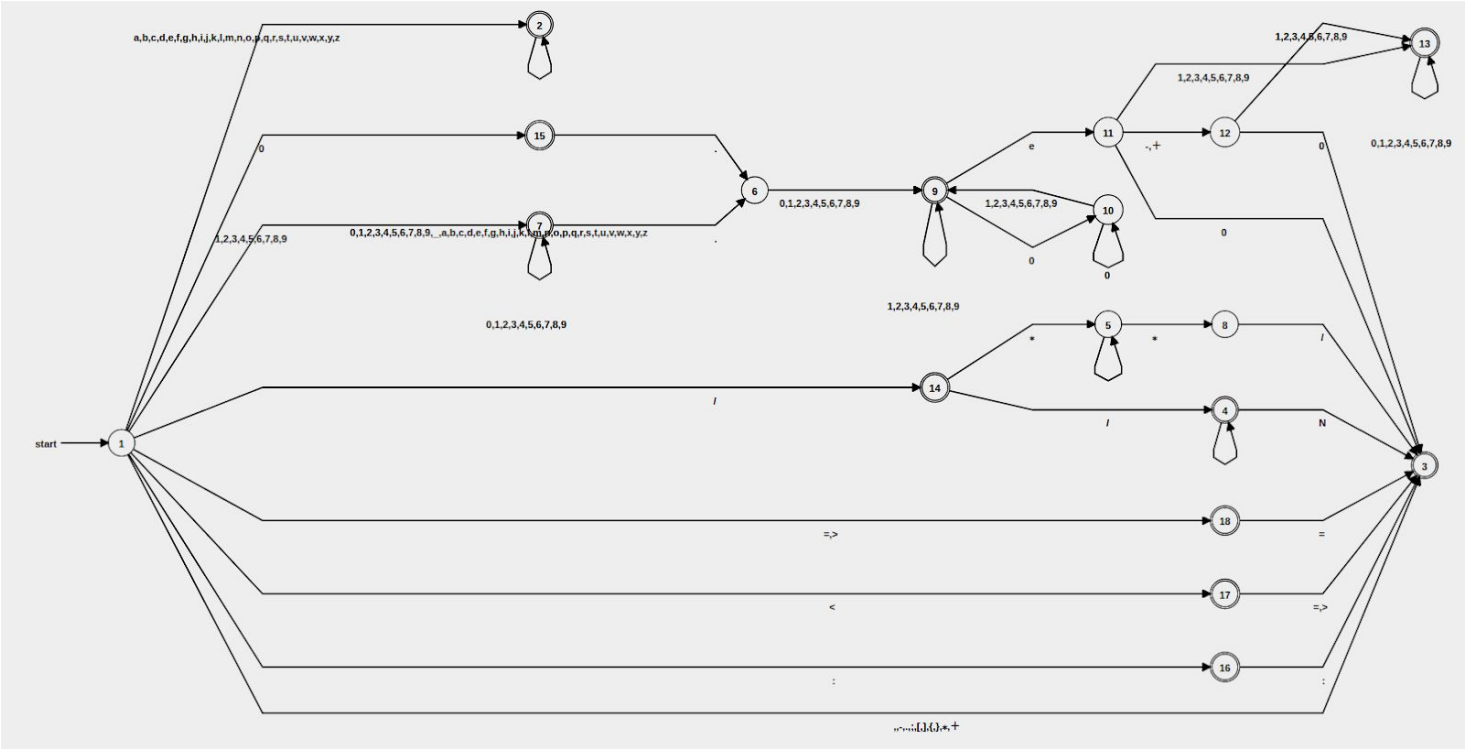

Figure 4: DFA representing *Token*

Section 3: Design

I implemented the lexical analyzer using the hand-written approach, making use of the State design pattern. The main components are the *Scanner*, and the 3 handlers *WordHandler, SymbolHandler and NumericalHandler*.

The *Scanner* provides an interface to read from a given stream and yield tokens found in the stream. It handles state related to the current lexeme, backtracking and location of the tokens in the source file. Upon reading a character from the source file, the *Scanner* instantiates the appropriate handler to simulate its corresponding DFA.

Handlers inherit from *CallableDFA*, which provides implementations for state transitions and yielding a token at a final state. Handlers store their current state as an attribute, and execute the given state's implementation (a Python function) every time a call is received. State transitions correspond to setting attribute to point to a different state.

Tokens are represented by a simple model class which stores the type, lexeme and location. Token Types are expressed as enum values, grouped by lexical similarity.

Section 4: Use of tools

I used the recommended CyberZHG's Toolbox to generate the DFAs from regular expressions.

The code base is written for Python 3.5+, with unit tests implemented using the built-in *unittest* package. No third party libraries are used in the execution of the lexical analyzer.