L-Università ta' Malta
Faculty of Information & Communication Technology

Assignment Report

Manwel Bugeja

September 17, 2020

Contents

1 Introduction **2**

2 Question 1 **2**
2.1 How the problem was tackled 2
2.2 Data Structures 2
2.3 FSA 2

3 Question 2 **2**
3.1 How the problem was tackled 2
3.1.1 AST Nodes 2
3.1.2 Parser 3

1 Introduction

This assignment is implemented in c++.

2 Question 1

2.1 How the problem was tackled

First off, an FSA was dedigned to accept numbers. Then the required structures where initialized in code. The FSA was translated to code in the form of transition table. Then, code was developed to successfully read the numbers with a stack and rollback capabilities. This was done according to the pseudo code provided in the text book. Following that, more and more elements where added, testing the lexer as it grew.

2.2 Data Structures

The following structures are defined in "transitions.h". Three enumerations were done to keep track of classifiers, states and token types. The enum for token types is divided into two parts (shown via the comment). The second part is used exclusivly by the parser.

An array of accepting states was declared to keep track of the states which were final. The transisition table was implemented as a 2d array with classifiers as columns and states as rows.

The char_cat and type tables were implemented as functions.

2.3 FSA

The following are parts of the FSA in the order they where added to the transition table.

3 Question 2

3.1 How the problem was tackled

3.1.1 AST Nodes

First off, the needed nodes were declared in ast.h. The nodes use inheritance to inherit features from more general nodes. These nodes hold data about what they represent, as well as pointers to their children nodes.

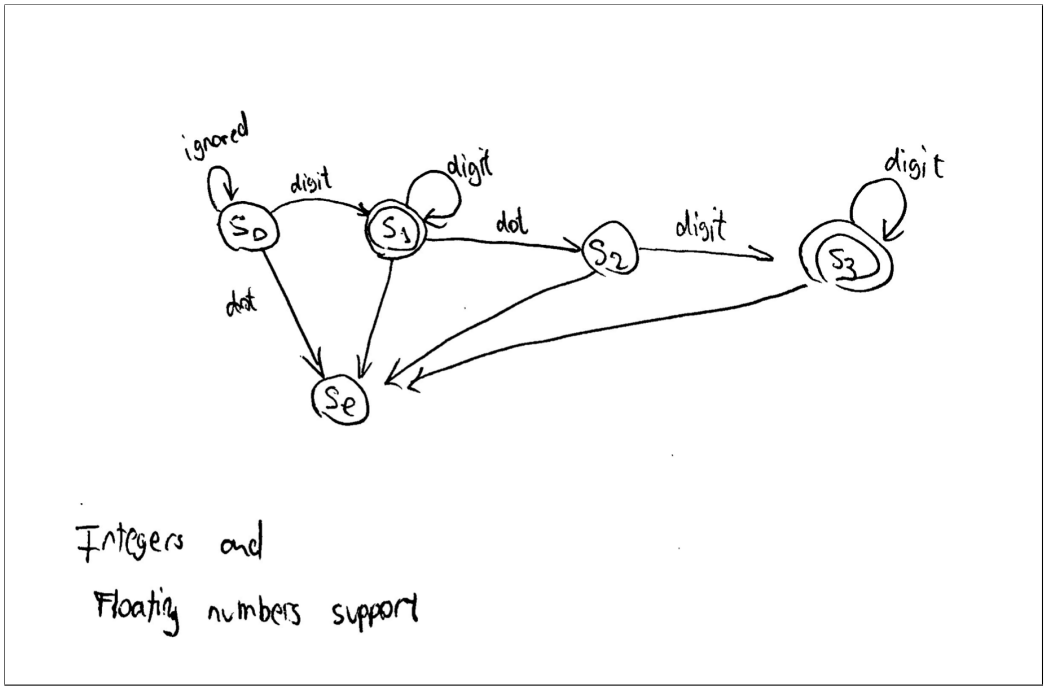

Figure 1: FSA (part concerned with numbers)

There are three abstract nodes. ASTNode is the super class which the other two inherit from. ASTStatementNode and ASTExpressionNode are the otehr two. All node classes inherit from these two. The only exception is ASTProgramNode, which is the topmost node of the Abstract Syntax Tree. This node consists of a list of statement nodes. This list is implemented as a vector.

3.1.2 Parser

The parser keeps track of the current token and the next token in line. This enables its look ahead functionalitiy thus classifying as an LL(1) parser. It works by using the current token and the next token to check if the structure of the input program is as valid, reporting errors if it is not.

In my implementation, functions such as parse_if_statement() were left out due to bad time managment from my end. However, these would have been very similar to the ones already implemented, as the concept is mostly the same.

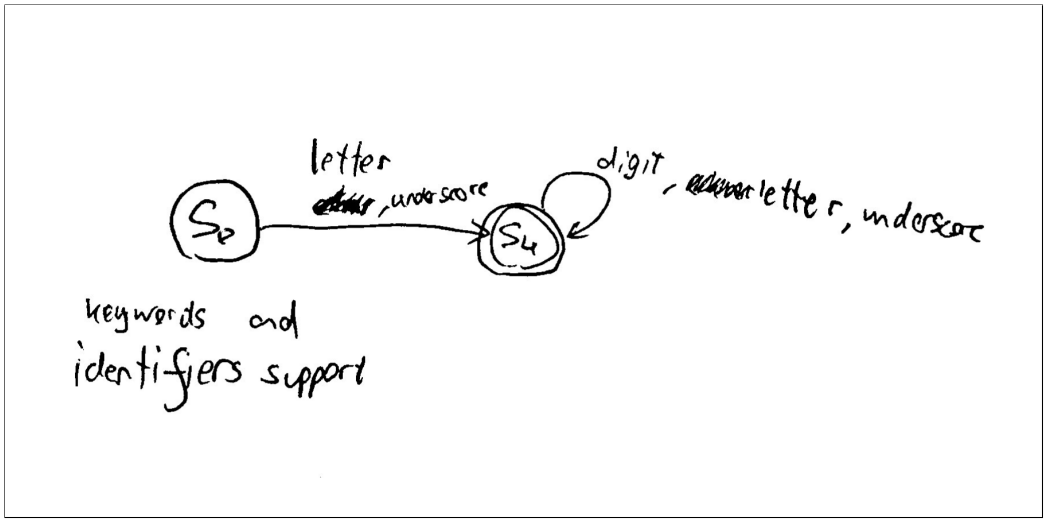

Figure 2: FSA (part concerned with identifiers and keywords)

4 Question 3

This question was not attampted.

5 Question 4

For this task, part of the Scope class was implemented. I would then have proceded to make the semantic analyser keep a stack of scopes.

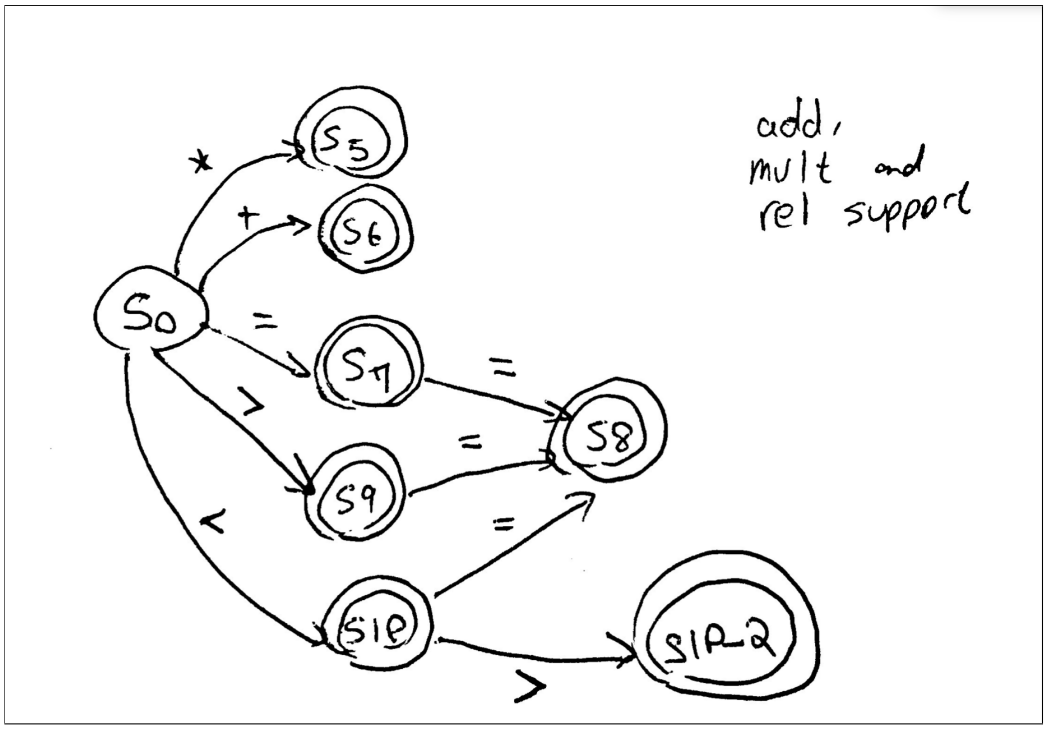

Figure 3: FSA (part concerned with operators)

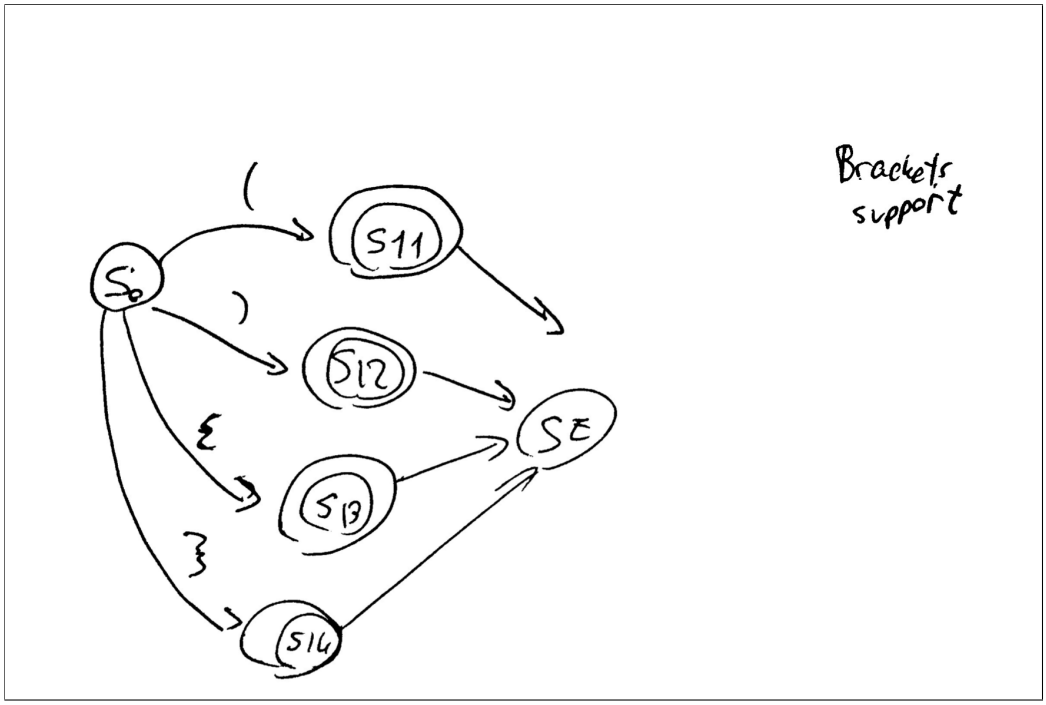

Figure 4: FSA (part concerned with brackets)

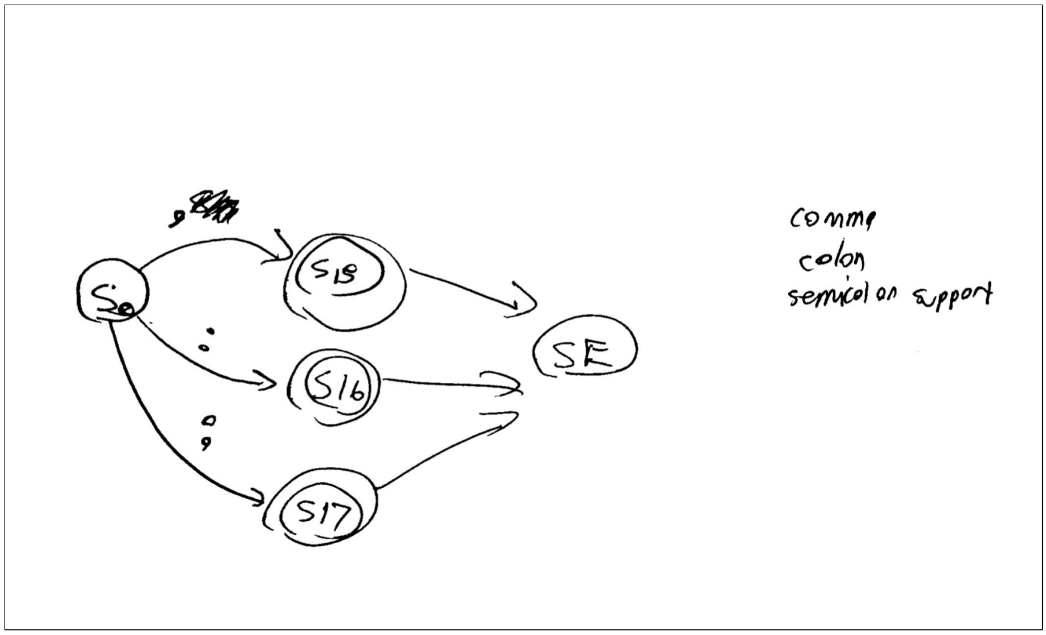

Figure 5: FSA (part concerned with colons and comma)

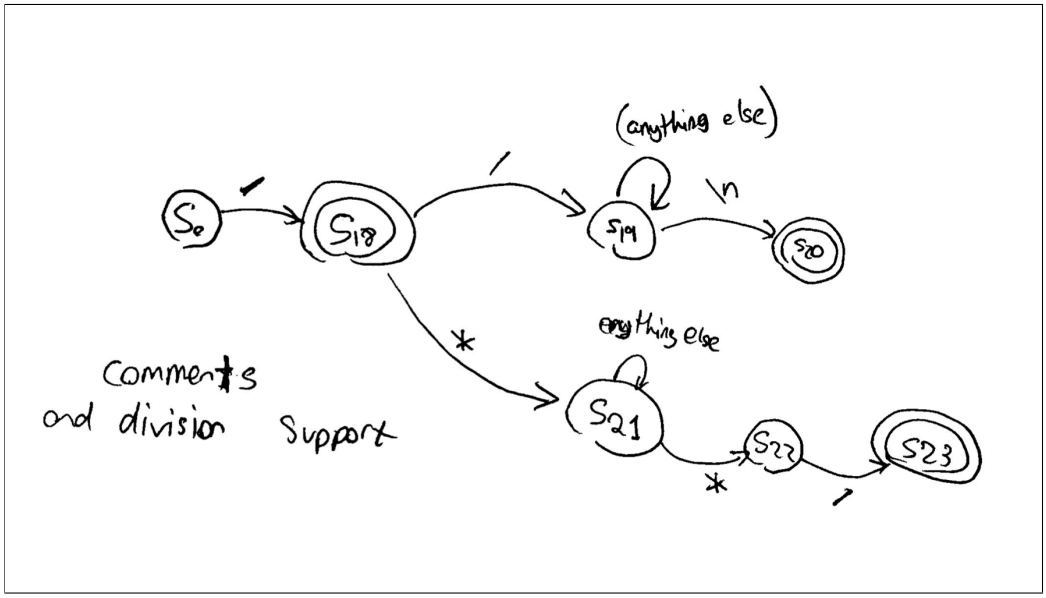

Figure 6: FSA (part concerned with comments and division operator