Finite State Machine: A Model for Validating Variants of Input Data

O.E. Oguike
Department of Computer Science
University of Nigeria
Nsukka, Nigeria
osonduoguike@unn.edu.ng

ABSTRACT
Finite state machine can be regarded as a form of state transition graph. It is a graph, whose nodes represent the states, while the edges represent the state transitions. It can be regarded as, a graph or network, whose nodes represent current value of state variables of software and the edges represent the state transitions or change of value of state variables, as a result of valid operations. In computing, finite state machine can be used to model different applications, both in hardware and software design. Any software application that has variables that can assume different values will define the states of the finite state machine. In each state, there are input and output, which represent state transitions into the state, and out of the state. The state transitions are as a result of different operations that cause the state variables to change values. This paper presents a novel way of using finite state machine to validate different types of input data in computer software.

Keywords- Finite state machine, input data format, data validation, state transition graph, spell checker, parser.

African Journal of Computing & ICT Reference Format:

1. INTRODUCTION

The need for input data validation in application software cannot be overemphasised, this is because data validation adds quality to computer software. Data validation helps to ensure that correct input data are used for data processing in an application software. The need for data validation has been the reason for the use of the slogan, “Gabbage In, Gabbage Out”. Most input data have defined format that can be used to generate a valid input data. Finite state machine can be used to model a particular input data format, which can be used to develop algorithm that can be implemented using a particular programming language. The implemented algorithm will detect correct or wrong input data. The development of computer program that will be used to validate input data is a complex and difficult task, but with the use of finite state machine, the complex task can be reduced to a simple one.

Similarly, spell checking can be regarded as a form of input data validation. The spelling of every word follows a defined format, which consists of the specific combination of letters that spells the word correctly. A spell checker will help to detect wrong spelling of a particular word. Most word processing software use spell checker to alert the user of wrong spelling of words. Finite state machine can be used to model a particular word or group of words, which can be used to develop algorithms that can be implemented as a computer program that will detect wrong spelling of words. Any implemented algorithm that performs this function is called a spell checker.

Another variant of input data that finite state machine can be used to validate is computer codes/programs. The process of validating computer codes is called parsing, and the software that is used to do parsing is called parser. The algorithm that can be implemented to become a parser can be developed using finite state machine. Such algorithm is developed by representing the syntax of the various vocabularies of a programming language as a finite state machine. Therefore, any computer code/program written in a particular programming language can be validated by determing if it conforms to the correct syntax.

2. SURVEY OF RELATED LITERATURE

Finite state machine is an important model that has been used to solve important problems in computing. In [1], finite state machine and fuzzy logic have been used for hand gesture learning and recognition, while [2] uses finite state machine and recurrence matrices to develop a secret sharing scheme for secure communication. In [3], FSM has been used to address the problem of encoding the state variable of a finite state machine such that the binary decision diagram representing the next state function and the output function have the minimum number of nodes. Though [4] has identified the use of finite state machine as a model that can be used to develop a spell checker, it emphasizes the use of javascript as a technology that can be used to accomplish this, which is a different approach from the approach that this paper uses.
This paper presents a novel way of developing finite state machine, which can be used to develop an algorithm that can be implemented using a programming language. In [5], the author presents a finite state machine pattern-matching algorithm that can be used for bioinformatics applications, while in [6], the author uses it to model and test web applications. On the other hand, [7] uses finite state machine to secure composite web service based on policy approach. The authors emphasize that finite state machine is an artificial intelligence technique, which has a mathematical root and can be used in matching pattern, sequential logic circuit, and implementing computer programs. Though [8] has identified that finite state machine can be used to validate input data, it has not identified other similar applications, like spell checker and parser, which this paper has considered. According to the author, a finite state machine is a collection of nodes (called states) with an arrow leaving each node for each possible input that can be encountered at that node. In [10], the authors present an illustration of what a finite state machine is. According to them, a machine is a system that can accept input, possibly produce output, which has some sort of internal memory that can keep track of certain information about inputs. They emphasize that the complete internal condition of the machine and all of its memory at any time is said to constitute the state of the machine at that time.

3. FSM AS A MODEL FOR VALIDATING INPUT DATA FORMAT

Codes, like vehicle registration number, course codes, GSM numbers can be assigned, using defined formats. It is, therefore necessary that validation checks be made any time users enter these codes on the computer. The validation check will help to sieve out wrong input data. Finite state machine can be used to represent input data format, with the aim of using it to develop an algorithm that can be implemented as a computer program that will validate input data based on the defined format. The following guidelines can be used to model an input data format using finite state machine.

- Start every finite state machine with a start state, i.e state, 0.
- Use a circle to represent each of the possible states, and number them.
- Use an arrow to link the various transitions of states, from a particular state.
- Label the arrows to indicate the value of the input character that will be read into the input variable or the value of the count or reset variable.
- You can group consecutive numeric character values; this will help to avoid too many states.
- Make sure that the identified states are distinct states.
- If an input character in the data format appears more than once in such a way that they follow each other, then the state will be treated as repeated state. In that situation, you can use an arrow that originate from a state to itself to indicate repetition of state. However, you need to use a count variable to indicate the number of times that the state will repeat itself.
  - If an input character in the data format appears more than once in such a way that they do not follow each other, then it will not be treated as repeated state. A reset variable will be used, which means that you are entering a state from two or more sources, and each source will lead to a different destination. You can use a reset variable to indicate the particular destination that a particular source will lead to.
  - Terminate every finite state machine with an end state, and number it appropriately.

The following examples will illustrate the use of the guidelines in producing the finite state machine for each of the data format shown below.

A. Course Code Data Format

The task is to develop a finite state machine that can be used to develop an algorithm, which can be implemented as a computer program that will check for valid/invalid input data format, using the course code data format, shown in figure 1. The course code data format begins with ‘CSC’, followed by digit space that can be any digit between 1 and 5. Following this is a 0 digit or digit that can be between 1 and 9. If it is a 0 digit, then the last digit will be any digit that is between 1 and 9, otherwise, the last digit will be any digit that is between 0 and 9.

<table>
<thead>
<tr>
<th>CSC</th>
<th>1-5</th>
<th>0</th>
<th>1-9</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-9</td>
<td>0-9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Course Code Data format
The guideline specified in section 3 can be used to develop the finite state machine as shown in figure 2.

B ELECTRONIC TIME DATA FORMAT

In the electronic time format shown in figure 3, the first digit space can be 0 – 1 digit or 2. If it is 0 – 1, then the next digit space will be 0 – 9, otherwise, it will be 0 – 3. Following the second digit space is a '.', which is followed in the third digit space with 0 – 5, and following this is 0 – 9 in the last digit space. The guideline outlined in section 3 can be used to produce the finite state machine for the electronic time data format. The finite state machine is shown in figure 4.

4. DEVELOPING ALGORITHM USING FSM

The algorithm, which can be implemented as a program, first initializes the state variable to zero, and it reads the input data to be validated one character at a time until it encounters the end of line character. Depending on the current value of the state variable, it calls the appropriate algorithm for that state. The algorithm for a particular state will check for the value of the character it has read, if it is the correct character for that state, it changes the state variable to the next state, otherwise it changes the state variable to error state. It continues this process until it reads the end of line character. In order to determine if the input data is correct or not, it checks for the value of the state variable.
If it is the ‘end’ state as specified in the finite state machine, then it means that the input data is correct; otherwise, the input data is wrong. The algorithms for the finite state machine shown in figure 2 and figure 4 follow.

A. Developing Algorithm From Course Code FSM

The algorithm for each of the states of the finite state machine shown in figure 2 has been outlined below:

```c
void zerotest(char data)
1. Read data
2. Determine zerotest
   2.1 IF data = ‘C’ THEN
   2.2 State := ‘1’;
   2.3 Ck := 1;
   2.4 State := ‘C’
   ELSE
   2.5 State := ‘E’
END

void twotest(char data)
1. Read data
2. Determine twotest
   2.1 IF data = ‘C’ THEN
   2.2 State := ‘1’;
   2.3 Ck := 2;
   2.4 State := ‘F’
   ELSE
   2.5 State := ‘E’
END

void ceetest(char data)
1. Read data
2. Determine ceetest
   2.1 IF data = ‘S’ THEN
   2.2 State := ‘2’
   ELSE
   2.3 State := ‘E’
END

void eeftest(char data)
1. Read data
2. Determine eeftest
   2.1 IF ch = ‘O’ THEN
   2.2 State := ‘6’
   ELSE
   2.3 IF data >= ‘1’ AND data <= ‘5’ THEN
   2.4 state := ‘3’;
   ELSE
   2.5 State := ‘E’
END
```

B. Developing Algorithm From Electronic Time FSM

In a similar manner, the algorithm for each of the states of the finite state machine shown in figure 4 has been outlined below:

```c
void threetest(char data)
1. Read data
2. Determine threetest
   2.1 IF data = ‘0’ THEN
   2.2 State := ‘6’
   ELSE
   2.3 IF data >= ‘1’ AND data <= ‘9’ THEN
   2.4 state := ‘7’;
   2.5 ct := 1;
   2.6 state := ‘8’
   ELSE
   2.7 State := ‘E’
END

void sixtest(char data)
1. Read data
2. Determine sixtest
   2.1 IF data >= ‘1’ AND data <= ‘9’ THEN
   2.2 state := ‘7’;
   2.3 ct := 2;
   2.4 state := ‘A’;
   2.5 state := ‘B’
   ELSE
   2.6 State := ‘E’
END

void eightest(char data)
1. Read data
2. Determine eightest
   2.1 IF data >= ‘0’ AND data <= ‘9’ THEN
   2.2 state := ‘9’;
   2.3 state := ‘B’
   ELSE
   2.4 State := ‘E’
END
```

```c
void zerotest(char data)
1. Read data
2. Determine zerotest
   2.1 IF data >= ‘0’ AND data <= ‘1’ THEN
   2.2 State := ‘1’;
   2.3 IF data = ‘2’ THEN
   2.4 state := ‘2’
   ELSE
   2.5 State := ‘E’
END
```
void onetest(char data)
1. Read data
2. Determine onetest
   2.1 IF data \geq '0' AND data \leq '9' THEN
   2.2 state := '4';
   2.3 ct := 1;
   2.4 state := '5'
   ELSE
   2.5 State := 'E'
END

void twotest(char data)
1. Read data
2. Determine twotest
   2.1 IF data \geq '0' AND data \leq '3' THEN
   2.2 state := '3'
   ELSE
   2.3 State := 'E'
END

void threetest(char data)
1. Read data
2. Determine threetest
   2.1 IF data = ':' THEN
   2.2 State = '6'
   ELSE
   2.3 state = 'E'
END

void fivetest(char data)
1. Read data
2. Determine fivetest
   2.1 IF ch = ':' THEN
   2.2 State := '6'
   ELSE
   2.3 State := 'E'
END

void sixtest(char data)
1. Read data
2. Determine sixtest
   2.1 IF data \geq '0' AND data \leq '5' THEN
   2.2 state := '7'
   ELSE
   2.3 State := 'E'
END

void seventest(char data)
1. Read data
2. Determine seventest
   2.1 IF data \geq '0' AND data \leq '9' THEN
   2.2 state := '4';
   2.3 ct := 2;
   2.4 state := '8';
   2.5 state := '9'
   ELSE
   2.6 State := 'E'
END

5. FSM AS A MODEL FOR DEVELOPING SPELL-CHECKER ALGORITHM

The same guideline outlined in section 3 can be used to develop a finite state machine, which can be used to develop an algorithm that can be implemented as a program that checks for the correct spelling of any word. Illustrating further, the finite state machine shown in figure 5 uses the guideline outlined in section 3, and it represents spelling of the word, ZIGZAG.

![Finite state machine for the word, ZIGZAG](image_url)
In a similar manner, the algorithm for each of the states of the finite state machine follows:

**Void zerotest(char data)**
1. Read data
2. Determine zerotest
   2.1 IF(data = 'Z') OR (data = 'z') THEN
   2.2 State := '1';
   2.3 zc := 1;
   2.4 State := '2'
   ELSE
   2.5 State := 'E'
END

**Void twotest(char data)**
1. Read data
2. Determine twotest
   2.1 IF (ch = '1') OR (ch = 'i') THEN
   2.2 State := '3'
   ELSE
   2.3 State := 'E'
END

**Void threetest(char data)**
1. Read data
2. Determine threetest
   2.1 IF (data = 'G') OR (data = 'g') THEN
   2.2 State := '4';
   2.3 gc := 1;
   2.4 state := '5'
   ELSE
   2.5 State := 'E'
END

**Void fivetest(char data)**
1. Read data
2. Determine fivetest
   2.1 IF(data = 'Z') OR (data = 'z') THEN
   2.2 State := '1';
   2.3 zc := 2;
   2.4 State := '6'
   ELSE
   2.5 State := 'E'
END

**Void sixtest(char data)**
1. Read data
2. Determine sixtest
   2.1 IF (data = 'A') OR (data = 'a') THEN
   2.2 State := '7'
   ELSE
   2.3 State := 'E'
END

**Void seventest(char data)**
1. Read data
2. Determine seventest
   2.1 IF (data = 'G') OR (data = 'g') THEN
   2.2 State := '4';
   2.3 gc := 2;
   2.4 state := '8';
   2.5 state := '9'
   ELSE
   2.6 State := 'E'
END

6. FSM AS A MODEL FOR DEVELOPING ALGORITHM FOR A PARSER

The syntax of a programming language, like Modula 2 can be expressed using EBNF or the syntax diagram. A finite state machine can be used to model the syntax of a particular vocabulary of a programming language, which can be used to develop an algorithm that can be implemented to become a parser. The syntax analyzer or parser will be used to verify if an input into the parser conforms to the syntax of the vocabulary of the language. The following example illustrate further. Figures 6a – 6d show the syntax of Modula 2 programming language as presented in [9]. Using the same guideline outlined in section 3, the syntax diagram for the vocabulary, integer can be represented as a finite state machine, shown in figure 7.
Figure 6: Syntax diagram of some Modula 2 vocabularies
The finite state machine shown in figure 7 can be used to develop algorithms that can be implemented to become a parser. The parser will check if the input to it is a valid or invalid integer number. The algorithm for each of the states of the finite state machine follows below:

**Void zerotest(char data)**
1. Request data
2. Determine zerotest
   2.1 IF data = octalDigit THEN
   2.2 state = ‘i’
   2.3 oc = oc + 1
   2.4 dc = dc + 1
   2.5 hd = hd + 1
   ELSE
   2.6 IF data = digit THEN
   2.7 state = ‘m’

**void mtest(char data)**
1. Request data
2. Determine mtest
   2.1 IF data = digit THEN
   2.2 dc = dc + 1
   2.3 hd = hd + 1
   ELSE
   2.4 IF dc < 5 THEN

Figure 7: Finite State Machine For The Modula-2 Vocabulary, Integer
void itest(char data)
1. Request data
2. Determine itest
   2.1 if data = octalDigit THEN
   2.2 oc = oc + 1
   2.3 dc = dc + 1
   2.4 hd = hd + 1
   2.5 if oc < 5 THEN
   2.6 state = 'i'
   2.7 ELSE
   2.8 state = 'E'
   2.9 ELSE
   2.10 state = 'E'
END

void ctest(char data)
1. Request data
2. Determine ctest
   2.1 if data = hexDigit THEN
   2.2 hd = hd + 1
   2.3 state = 'd'
   2.4 ELSE
   2.5 if data = 'H' THEN
   2.6 state = 'g'
   2.7 state = 'E'
END

void dtest(char data)
1. Request data
2. Determine dtest
   2.1 if data = hexDigit THEN
   2.2 hd = hd + 1
   2.3 ELSE
   2.4 if hd < 5 THEN
   2.5 state = 'd'
   2.6 ELSE
   2.7 state = 'f'
   2.8 state = 'E'
END

void ftest(char data)
1. Request data
2. Determine ftest
   2.1 if data = 'H' THEN
   2.2 state = 'g'
   2.3 state = 'l'
   2.4 ELSE
   2.5 state = 'E'
END

void jtest(char data)
1. Request data
2. Determine jtest
   2.1 if data = 'B' THEN
   2.2 state = 'h'
   2.3 state = 'l'
   2.4 ELSE
   2.5 state = 'k'
   2.6 state = 'l'
   2.7 state = 'E'
END
7. IMPLEMENTATIONS OF THE ALGORITHMS

Pascal programming languages was used to implement the algorithms. Each of the algorithms for a state of the finite state machine was implemented, as a Pascal procedure, which does not return any value. A test program was written, which requested for the input data, which was read one character at a time until an end of line character was encountered. After reading each character of the input data, it checked for the current state of the state variable and called the appropriate procedure for that state.

8. CONCLUSION

This paper has been able to use finite state machine to model three variants of input data. It has defined algorithms for each finite state machine that has represented a particular variant of input data, the algorithms have been implemented using Pascal programming languages, and test programs have been written to validate each variant of input data.

REFERENCES


Author’s Brief

Oguike, Osondu Everestus is a Senior Lecturer in the Department of Computer Science, University of Nigeria, Nsukka, Enugu State, Nigeria. He has received many academic prizes and scholarships as a result of his outstanding academic performance. He is interested in modeling the performance of parallel computer system. He can be reached by phone on +2348035405100 and through E-mail osondu.oguike@unn.edu.ng