Digital Innovations
& Contemporary Research in
SCIENCE & ENGINEERING

The Journal of Digital Innovations & Contemporary Research in Science & Engineering is an
International Journal of
The Multidisciplinary Research Group
International Centre for IT & Development, USA

Article Citation Format
E. E. Ogheneovo (2016) An Alternative Algorithm for Simulating Finite-State Automata. Journal of Digital Innovations & Contemp Res. In Sc., & Eng. Vol. 4, No. 4. Pp 71-82

Article Progress Time Stamps

Article Type: Research Article
Manuscript Received: 11th October, 2016
Review Type: Blind
Word Count Post Review: 3658
Review/Acceptance Information Sent : 17th Dec, 2016
Final Acceptance:: 27th Dec. 2016
DOI Prefix: 10.22624

An Alternative Algorithm for Simulating Finite-State Automata

E. E. Ogheneovo
Department of Computer Science
University of Port Harcourt
Port Harcourt, Nigeria
E-mail: edward_ogheneovo@yahoo.com

ABSTRACT

In this paper, an alternative algorithm for simulating the finite-state automaton is developed. This algorithm simulates the finite-state automaton by repeated application of the next-state function. This is done in each case where the outputs strings ends in 011e, 100e, and 110e respectively for input strings of 0s and 1s with end marker e. The states s_o, s_1, s_2, and s_3 are denoted by 0, 1, 2, and 3 respectively. The use of the finite-state automaton allows the creator of the algorithm to focus on each step of the analysis of the input string independently of the other steps. For each algorithm, there is an alternative algorithm that simulates the finite-state automaton by repeated application of the next-state function. These are done by determining and entering the values of the next-state function directly as a two-dimensional array. These results show that it is possible to simulate a finite-state automaton using a generic algorithm to determine the value of the next-state of a finite automaton.

Keywords: Algorithm, Finite-State Automaton, Simulation, Turing machine, automata

1. INTRODUCTION

A finite-state automaton [1] [2] [3] [4] is a primitive but useful computational model for both hardware and software. They are used to model system behavior in both engineering and scientific applications [5]. It is found very useful in the field of computer science, mathematics, physics, computer engineering, electrical engineering, electronic engineering and linguistics. It is a mathematical model of computation used for designing both computer programs and sequential logic circuits. It is a state machine where Σ is a finite set. With finite-state automaton, a system's behavior can be modeled as a set of states and rules that govern transition between them. The machine provides a simple computational model in the computer science, physics, engineering and linguistics [6] [7]. It is similar to Turing machines [8] [9] [10] in that it has a controller with a movable read/write head except that in finite-state automata, the head moves in only one direction unlike in Turing machine, where the head can move in both directions, left or right [11] [12]. Just as in Turing machine, the sequence of symbols being read constitutes the input symbols; while the sequence of symbols being written constitute the output.

A finite-state automaton consists of a set of states (e.g., s_o, s_1, s_2, ...) together with an indication of which is the initial state and which are the accepting states, a list of all input symbols, and specification for next-state function that defines which state is produced by each input in each state [13] [14]. It is a state machine in which a simple model of computation can be constructed that represents the abstract state of a machine. The finite-state automata are much more restrictive in their computational capabilities and usage when compared to Turing machines since the head can only move in one direction. For example, simpler languages such as the sequence of well-balanced parenthesis strings cannot be recognized by finite-state automata.

A number of works have been proposed for simulating finite-state automaton using Java[15], jFAST[[16], and jFLAP [18] [19] [20] [21]; some of these software are actually working but are Java dependent. However, in this paper, we proposed a generic algorithm and its alternative that can be used by any programming language for simulating finite-state automata. This way, a programmer is not limited to a particular programming language to be able to develop or use a simulator to simulate finite-sate automata especially when the subject matter of the concept and the theory of computation are quiet challenging and difficult for students of the subject to comprehend easily. Thus our contribution to this paper is to ensure that students of automata and computational theory in general find the subject much easier to understand and comprehend as opposed to the present situation where students find the subject very dreadful. It is hoped that these alternative algorithms will be implemented by interested researchers using different programming languages to provide alternative ways of solving finite automata problems and computational theory in general.

2. STATEMENT OF THE PROBLEM

The finite-state automata are much more restrictive in their computational capabilities and usage when compared to Turing machines and other computation models since the head can only move in one direction. The use of the finite-state automaton allows the creator of the algorithm to focus on each step of the analysis of the input string independently of the other steps. A number of works have been proposed for simulating finite-state automaton using Java, jFAST, and FLAP; some of these software are actually working but are Java dependent. However, in this paper, we proposed a generic algorithm that can be used by any programming language for simulating finite-state automata. This way, a programmer is not limited to a particular programming language to be able to develop or use a simulator to simulate finite-sate automata especially when the subject matter of the concept and the theory of computation are quiet challenging and difficult for students of the subject to comprehend easily. This paper proposes a generic algorithm for simulating the finite-state automaton. For each set of algorithm, an alternative algorithm that simulates the finite-state automaton by repeated application of the next-state function.

3. METHODOLOGY

An algorithm that simulates the finite-state automaton by repeated application of the next-state function is developed. The states s_0, s_1, s_2, and s_3 are denoted by 0, 1, 2, and 3 respectively. For each algorithm developed an equivalent alternative algorithm that uses software is also developed to ensure that the software can actually be used in place of the algorithm.

3.1 Algorithm for Simulating Finite-State Automaton with Output String that ends in 011e

The first algorithm developed consist of input strings 0s and 1s with an end marker indicating that the strings have been exhausted and the resulting output consists of string that ends in 011e. The algorithm is shown in figure 1. The alternative algorithm to this is shown as algorithm 2 in figure 2.

Algorithm 1: An algorithm that simulate the finite-state automaton by repeated application of the next-state function. The states s_o, s_1, s_2, and s_3 are denoted by 0, 1, 2, and 3 respectively

```
Input: string of 0s and 1s with an end marker e
Output: string that ends in 011e
begin // the beginning of the body of the algorithm

   state := 0
   symbol := 1 // first symbol in the input string

  while (symbol ≠ e) // the while loop begins here
        if state = 0 then
            if symbol = 0 then state := 1
            else state := 0
        else if state = 1 then
       if symbol = 0 then state := 1
       else state := 2
       else if state = 2 then
        if symbol = 0 then state := 1
         else state := 3
        else if state = 3 then
        if symbol = 0 then state := 1
         else state := 0
            symbol := next symbol in the input string
    end while // end the while loop
end // end the begin block
```

Fig. 1: An algorithm that simulates a finite-state automaton with strings that ends in 011e

As seen in algorithm 1, if the symbols in the strings are not exhausted and at state s_0, reading input symbols 0, the transition moves to the next-state, s_1, otherwise it stay put at state s_0. If in state s_1, and input symbol is 0, then it stays put in s_1 otherwise it transit to state s_2. At state s_2, if the input symbol is 0, it transit back to state s_1, otherwise it transit to state s_3. At state s_3, the string is accepted. Finally, at s_3, if the input symbol is 0, it transit back to state s_1, otherwise, it transit back to state s_0.

3.1.1 Alternative Algorithms for Simulating Finite-State Automaton with Output String that ends in 011e

Alternatively, it is possible to simulate a finite-state automaton using a program by entering the values of the next-state function directly in a two-dimensional array. We then develop an alternative algorithm for this simulation method using the finite-state automata in figure for algorithm 2. These algorithms are described as algorithms 2.

Algorithm 2: An alternative algorithm that simulate the finite-state automaton by repeated application of the next-state function. The states s_0, s_1, s_2, and s_3 are denoted by 0, 1, 2, and 3 respectively

```
Input: string of 0s and 1s with an end marker e
Output: string that ends in 011e
        Let N(x, y) be a function where x and y are the state and input symbol respectively. Thus for
        the finite-state automata in figure 3
                N(0, 0) := 1;
                N(0, 1) := 0;
                N(1, 0) := 1;
                N(1, 1) := 2;
                N(2, 0) := 1;
                N(2, 1) := 3;
                N(3, 0) := 1;
                N(3, 1) := 0.

Begin
        state := 0
        input symbol := first symbol in the string
        while (input symbol ≠ e) // begin of while loop
                state := N(state, input symbol)
                input symbol := next symbol in the input string
        end while // end of while loop
end. // end the begin block
```

Fig. 2: An alternative algorithm that simulates a finite-state automaton with strings that ends in 011e

Using the same technique, the automaton in figure 3 recognizes strings that end in 011.

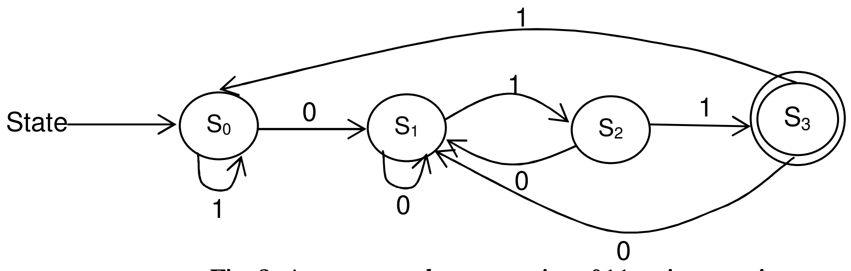

Fig. 3: Automaton that recognizes 011 as input string

Based on figure 3, the annotated next-state table for input string 011 is shown in Table 1.

Table 1: Next-state table for input string 011

State	0	1
S_0	S_1	S_0
S_1	S_1	S_2
S_2	S_1	S_3
S_3	S_1	S_0

3.2 Algorithm for Simulating Finite-State Automaton with Output String that ends in 100e

Using the same procedure,, it is possible to have a similar algorithm if we initialize the starting state to 0 but this time around, we set the first input symbol to 1. The algorithm is shown as algorithm 3 for the input string that ends in 110 using the finite-state automaton in figure 4.

Algorithm 3: An alternative algorithm that simulate the finite-state automaton by repeated application of the next-state function. The states s_o, s_1, s_2, and s_3 are denoted by 0, 1, 2, and 3 respectively

```
Input: string of 0s and 1s with an end marker e
Output: string that ends in 100e
begin // the beginning of the body of the algorithm

    state := 0
    symbol := 1 // first symbol in the input string

    while (symbol ≠ e) // the while loop begins here

    if state = 0            then if symbol = 0
                                 then state := 0
                                 else state := 1

    else if state = 1   then if symbol = 0
                                 then state := 2
                                 else state := 1

    else if state = 2   then if symbol = 0
                                 then state := 3
                                 else state := 1

    else if state = 3   then if symbol = 0
                                 then state := 1
                                 else state := 0

    symbol := next symbol in the input string

    end while // end the while loop

end. // end the begin block
```

Fig. 4: An algorithm that simulates a finite-state automaton with strings that ends in 100e

3.2.1 An Alternate Algorithm for Simulating Finite-State Automaton with Output String that ends in 100e

An Alternate Algorithm for simulating finite-state automaton with output string that ends in 100e is shown in figure 5.

```
Algorithm 4: An alternative algorithm that simulate the finite-state automaton by repeated application
             of the next-state function. The states s₀, s₁, s₂, and s₃ are denoted by 0, 1, 2, and 3
             respectively.
Input: string of 0s and 1s with an end marker e
Output: string that ends in 100e
        Let N(x, y) be a function where x and y are the state and input symbol respectively. Thus for
        the finite-state automata in figure ...,
                N(0, 0) := 0;
                N(0, 1) := 1;
                N(1, 0) := 2;
                N(1, 1) := 1;
                N(2, 0) := 3;
                N(2, 1) := 1;
                N(3, 0) := 1;
                N(3, 1) := 0.
begin
        state  := 0
        input symbol := first symbol in the string
        while (input symbol ≠ e) // begin of while loop
state := N(state, input symbol)
                input symbol := next symbol in the input string
        end while // end of while loop
end. // end the begin block
```

Fig. 5: An algorithm that simulates a finite-state automaton with strings that ends in 100e

In figure 6, the finite automaton shown is a next-state automaton that recognizes the string that ends in 100.

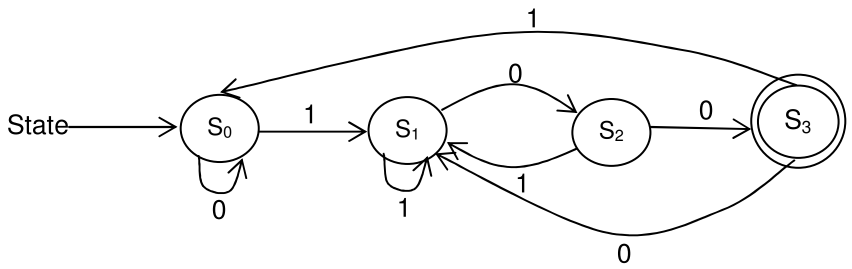

Fig.6: Finite-State automaton that end in 100

The automaton in figure 6 recognizes strings that end in 100. Starting with input symbol 1, the next-state is determined. Table 3 the annotated output table for next-state string that recognizes input string 100.

Table 3: Next-state table for input string 100

State	0	1
s_0	s_0	s_1
s_1	s_2	s_1
s_2	s_3	s_1
s_3	s_1	s_0

3.3 Algorithm for Simulating Finite-State Automaton with Output String that ends in 110e

In the same way, a similar algorithm is used to simulate a finite-state automaton by initializing the starting state to 0 but this time around, a different input string is used. The algorithm is shown as algorithm 5 for the input string that ends in 110 using the finite-state automaton in figure 6.

Algorithm 5: An algorithm that simulate the finite-state automaton by repeated application of the next-state function. The states s_o, s_1, s_2, and s_3 are denoted by 0, 1, 2, and 3 respectively

```
Input: string of 0s and 1s with an end marker e
Output: string that ends in 110e
begin // the beginning of the body of the algorithm

        state := 0
        symbol := 1 // first symbol in the input string

        while (symbol ≠ e) // the while loop begins here

        if state = 0               then if symbol := 0
                                           then state := 0
                                           else state := 1

        else if state = 1    then if symbol = 0
                                           then state := 1
                                           else state := 2

        else if state = 2    then if symbol = 0
                                           then state := 3
                                           else state := 1

        else if state = 3    then if symbol = 0
                                           then state := 1
                                           else state := 0

        symbol := next symbol in the input string

        end while // end the while loop

end. // end the begin block
```

Fig. 7: An algorithm that simulates a finite-state automaton with strings that ends in 110e

3.3.1 An Alternate Algorithm for Simulating Finite-State Automaton with Output String that ends in 110e

An Alternate Algorithm for simulating finite-state automaton with output string that ends in 110e is shown in figure 7.

Algorithm 6: An alternative algorithm that simulate the finite-state automaton by repeated application of the next-state function. The states s_0, s_1, s_2, and s_3 are denoted by 0, 1, 2, and 3 respectively

Input: string of 0s and 1s with an end marker e
Output: string that ends in **110e**

```
        Let N(x, y) be a function where x and y are the state and input symbol respectively. Thus for
        the finite-state automata in figure ...,
                N(0, 0) := 0;
                N(0, 1) := 1;
                N(1, 0) := 1;
                N(1, 1) := 2;
                N(2, 0) := 3;
                N(2, 1) := 1;
                N(3, 0) := 1;
                N(3, 1) := 0.

begin
        state := 0
        input symbol := first symbol in the string
        while (input symbol ≠ e) // begin of while loop
                state := N(state, input symbol)
                input symbol := next symbol in the input string
        end while // end of while loop
end. // end the begin block
```

Fig. 8: An alternative algorithm that simulates a finite-state automaton with strings that ends in 110e

Figure 9 is a next-state automaton that recognizes the string that ends in 110.

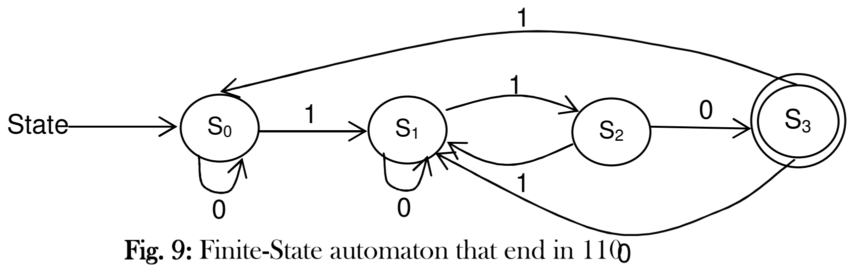

Fig. 9: Finite-State automaton that end in 110

Figure 9 is the annotated next-state table 3 for input string 110.

Table 3: Next-state table for input string 110

State	0	1
S_0	S_0	S_1
S_1	S_1	S_2
S_2	S_3	S_1
S_3	S_1	S_0

4. RESULTS AND DISCUSSION

Using the automata in figures 3, 6, and 9, results are generated for next-state input strings 011, 100, and 110 in tables 4, 5, and 6 respectively. Table 4 shows the output of the next state input string 011 for the alternative algorithm.

Table 4: Next-state table for 011 using the alternate algorithm

State	0	1
S_0	0	1
S_1	2	1
S_2	3	1
S_3	0	1

The first algorithm developed consist of input strings 0s and 1s with an end marker indication that the strings have been exhausted and the resulting output consists of string that that ends in 100e. The algorithm is shown in figure 4. The alternative algorithm, algorithm 3 is shown in figure 5. The automaton for this algorithm is shown in figure 6. Next-state table for 100 using the alternate algorithm using the alternate algorithm

Table 5: Next-state table for 100 using the alternate algorithm using the alternate algorithm

State	0	1
S_0	0	1
S_1	2	1
S_2	3	1
S_3	1	0

The results of algorithm 5 in figure 7 and the alternative algorithm 6 in figure 8 are actually the same. This shows that an algorithm that simulates a finite-state automaton with strings that ends in 011e can also used an alternative algorithm to simulate the same automaton using the same string value. Table 6 shows the next-state result for input string 110

Table 6: Next-state table for input string 110 using the alternate algorithm using the alternate algorithm

State	0	1
S_0	0	1
S_1	1	2
S_2	3	1
S_3	1	0

5. CONCLUSION

An alternative algorithm for simulating the finite-state automaton is proposed in this paper. An algorithm that simulates the finite-state automaton by repeated application of the next-state function is developed. The states s_0, s_1, s_2, and s_3 are denoted by 0, 1, 2, and 3 respectively. For each algorithm developed an equivalent alternative algorithm that uses software is also developed to ensure that the software can actually be used in place of the algorithm. The use of the finite-state automaton allows the creator of the algorithm to focus on each step of the analysis of the input string independently of the other steps. For each algorithm, an alternative algorithm that simulates the finite-state automaton by repeated application of the next-state function. These are done by determining and entering the values of the next-state function directly as a two-dimensional array. This algorithm simulates the finite-state automaton by repeated application of the next-state function. This is done in each case where the outputs strings ends in 011e, 100e, and 110e respectively for input strings of 0s and 1s with end marker e. These results show that it is possible to simulate a finite-state automaton using algorithms determine the value of the next-state of a finite automaton

REFERENCES

[1] Protelemaeus, C. (2014). System Design, Modeling and Simulation Using Ptolemy II, First Edition, Version 1.0, http://ptolemy.eecs.berkeley.edu/books/Systems/chapters/FiniteStateMachines.pdf.

[2] Calcude,C., Calcude, E., and Khiussainov, B. (1997). Deterministic Automata: Simulation, Universality, and Minimality. Annals of Pure and Applied Logic 90, pp. 263-276.

[3] Grib, A. A. and Zapatrin, R. R. (1990). Automata Simulating Quantum Logics. Int'l Journal of Theoretical Physics, Vol. 29, No. 2, pp. 113-123.

[4] Hopcroft, J. E. and Ullman, J. D. **(1979).** Introduction to Automata Theory, Languages, and Computation.Addison-Wesley, Reading, MA.

[5] Pighizzi, G. (2012). Two-way Finite Automata: Old and Recent Results in E. Formenti (Ed.): AUTOMATA and JAC 2012 Conferences, EPTCS 90, 2012, pp. 3-20. Doi: 10.4204/EPTCS.90.1.

[6] Geffert, V., Guillon, B. and Pighizzini, G. (2012). Two-way Automata Making Choices only at the Endmarkers. In Adrian Horia Dediu and Carlos Martin-Vide, Editors: LATA, Lecture Notes in Computer Science 7183, Springer, pp. 264-276. Doi:10.1007/978-3-642-28332-1.23.

[7] Chroback, M. (2003). Errata to: Finite Automata and Unary Languages. Theoretical Computer Science 302(1-3), pp. 497-498. Doi: 10.1016/S0304-3975(03)00136-1.

[8] Calude, C. S. and Lipponen, M. (1998). Computational Complementarily and Sofic Shift in X. Lin (ed.). Theory of Computing. In Proceedings of the 4th Australasian Theory Symposium, CATS'98, Springer-Verlag, Singapore, pp. 277-290.

[9] Dvurečenskij, A., Pulmannova, S. and Svozil, K. (1995). Partition Logics, Orthoalgebras and Automata. Helvetica Physical **Acta** 68, pp. 407-428.

[10] Mallozz, J. S. and Delillo, N. J. (1984). Computability with Pascal, Englewood Cliffs, NJ: Prentice-Hall,Inc.

[11] Kopeček,I. and Škarvada, L. (2003). Modeling Dialogue Systems by finite Automata, FIMU Report Series, FIMU-RS2003-01, faculty of Informatics, Masaryk University.

[12] Black, A. P. and Tolmach, A. (2003). Nondeterministic Finite-State Automata Lecture Note. http://web.cecs.pdx.edu/~black/CS311/Lecture%20Notes/Lecture%2003.pdf.

[13] Balcerzak, M. and Niwinski, D. (2010). Two-way Deterministic Automata with Two Reversals are Exponentially More Succinct than with One Reversal. Information Processing Lett. 110(10), pp. 346-398. Doi:10.1016/j.ipl.2010.03.008.

[14] Chrobak, M. (1986). Finite Automata and Unary Languages. Theoretical Computer Science, 4793), pp. 149-158.

[15] White, T. M. and Way, T. P. (2006). jFAST: A Java Finite Automata Simulator. In Proceedings of the ACM SIGCSE'06, March 1-5, 2006, Houston, Texas, USA.

[16] DeLillo,N. J. (2008). Simulations of Finite-State Automata Using Java 6.0. Part One: Preliminaries, Technical Report Number 256, November 2008, Ivan G. Seidenberg School of Computer Science and Information Systems, Pace University. http://support.csis.pace.edu/CSISWeb/docs/techReports/techReport260.pdf

[17] Bovet, J. (2004). Visual Automata Simulator: A Tool for Simulating Automata and Turing Machines. University of San Francisco. http:www.cs.usfca.edu/~jbovet/vas.html.2004.

[18] Cavaleante, R., Finley, T. and Rogers, S. H. (2004). A Visual and Interactive Automata Theory Course with JFLAP 4.0. In ACM SIGCSE'04 Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education, March 3-7, 2004, Norfolk, Virginia, USA, pp. 140-144. Doi: 10.1145/971300.971347.

Digital Innovations
& Contemporary Research in
SCIENCE & ENGINEERING
Vol. 4 No. 4, Dec. 2016

[19] Rodger, S. H. (2009). Increasing Engagement in Automata Theory with JFLAP, Visual Thinking Workshop, SigCSE'09, March 1, 2009, Duke University.

[20] Koza, J. R., Goldberg, D. E., Finley, T. and Roger, S. H.(2004). A Visual and Interactive Automata Theory Course with JFLAP 4.0. In Proceedings of SIGCSE'04, Vol. 36, No. 1, pp. 140-144, ACM Press.

[21] Rogers, S.H. and Gramond, E. (1998). JFLAP: An Aid to Studying Theories in Automata Theory. In SIGCSE Bulletin inroads. Proceedings of ITiCSE'98, Vol. 37, No. 3, pp. 325-329, ACM Press.