Journal of Advances in Mathematics and Computer Science

29(5): 1-10, 2018; Article no.JAMCS.37848
ISSN: 2456-9968
(Past name: British Journal of Mathematics & Computer Science, Past ISSN: 2231-0851)

A New Algorithm for Determining the Equivalence of Two Finite-State Automata

Edward E. Ogheneovo[1*]

[1]Department of Computer Science, University of Port Harcourt, Port Harcourt, Nigeria.

Author's contribution

The sole author designed, analyzed and interpreted and prepared the manuscript.

Article Information

DOI: 10.9734/JAMCS/2018/37848
Editor(s):
(1) Dr. Octav Olteanu, Professor, Department of Mathematics-Informatics, University Politehnica of Bucharest, Bucharest, Romania.
Reviewers:
(1) T. Marimuthu, Ayya Nadar Janaki Ammal College, India.
(2) Sangeeta Ahuja, Delhi University, Senior Scientist IASRI (ICAR), India.
(3) Udoinyang G. Inyang, University of Uyo, Nigeria.
Complete Peer review History: http://www.sciencedomain.org/review-history/27526

Original Research Article

Received: 18 September 2017
Accepted: 21 November 2017
Published: 01 December 2018

Abstract

Finite-state automaton is a machine that processes input strings and produces output indicating whether the input string is accepted or not. It is an acceptor recognizer for input specification. A finite-state automaton is an input/output device that accepts strings as input and produces binary numbers 0s and 1s. Two automata are equivalent if they generate the same or similar output for each input string. That is to say, two automata are equivalent if and only if they have the same computing powers. In this paper, we develop an algorithm that can be used to determine if two automata are equivalent. Such automaton could be an non-deterministic finite automata (NFA) that is converted to deterministic finite automata (DFA) or a DFA that is minimized into another DFA (minimized DFA) which are equivalent in the sense that they have the same computing power and can therefore be used to compute the same regular expression. Examples of the use of the algorithms are provided and their results show that they are equivalent in all respects. From the examples, it is clearly seen that each pair of automaton accept the same language, hence they are said to be equivalent. The proposed algorithm performs better in terms of time and space complexities when compared with existing algorithms because it runs faster and occupies less space in the computer's memory.

Keywords: Algorithm; equivalent; finite automata; language; computational model; computing.

**Corresponding author: E-mail: edward_ogheneovo@yahoo.com;*

1 Introduction

Finite automata are models of computation used for computing both hardware and software [1]. They are computational models that have applicability in many fields of study such as computer science, engineering, mathematics, and linguistics [2,3]. Finite automaton is similar to Turing machines except that its head moves in only one direction, whereas the Turing machine can rotate in directions, left or right. They are more restrictive in their capabilities and usage when compared to Turing machines. Therefore, we can describe a finite automaton as an acceptor recognizer for language specification. It is often used for lexical analysis when developing compilers. Positions in finite automata are represented with circles; these positions are referred to as states. The states are joined together using arrows that point from one state to the next; these arrows are called edges. These edges are labeled with the input symbols or an empty string where there is no input symbol. A finite automaton with multiple states can have output devices attached to each state so that the automaton can classify input stings into different categories, one for each accepting state [4,5].

A finite automaton [6] can be constructed by using a number of states which usually starts with the initial state and ends with one or final or accepting states, some input symbols, and specification for a next-state function. To construct an automaton, the transition diagram is used. It is the transition diagram that explains the structure of the automata by the edges with arrows which indicates the direction of flow of the transition diagram from one state to the other. These arrows are labeled with input symbols which points from one state to the other. The starting point is called the initial state and then moves on until it gets to the accepting state(s). Usually, the states are drawn using circles and final or accepting state(s) is represented by double circles to indicate that the automata has reached the terminal stage [7,8]. Two automata are equivalent if they generate the same or similar output for each input string [9,10].

This paper proposes an algorithm that can be used to simulate two equivalent finite state automata. Often times, the automata that are equivalent could be an NFA that is converted to DFA or a DFA that is minimized or reduced to a smaller DFA with reduced number of states. A nondeterministic finite automaton transformed into a deterministic finite automaton having fewer states when compared with the original NFA. The two automata are said to be equivalent if they have the same power and can therefore be used to compute the same regular expression. Examples of the use of the algorithm are provided and their results show that they are equivalent in all respects. The two automata that are produced are equivalent if they produce identical output for each input string or if they have the same computing power by accepting and processing the same language.

2 Describing Finite-State Automata

A finite automaton [11] is an input/output device whose input consists of strings or labels and whose output consists of 0s and 1s. It is a machine that is used to process input strings and provides output showing whether the input string is accepted or not [12,13]. Informally, a finite automaton is a computational machine used for computation and for recognizing regular expressions [14].

Definition 1 (Finite State Automata): Formally, we define an automaton as a quintuple $\mathcal{D} = (Q, \Sigma, \delta, q_o, F)$, where Q is a finite set of states, Σ represents the input symbols called alphabets, $\delta : Q \times \Sigma \rightarrow Q$ = a transition function, q_o is a start state ($q_0 \in Q$), and $F \subseteq S$ = a set of accepting states.

Two types of automata exist. These are: Finite automata can be deterministic or non-deterministic.

2.1 Nondeterministic finite automata (NFA)

An NFA is a quintuple $N = (Q, \Sigma, \delta, q_o, F)$, where Q is a finite set of states, Σ is of input symbols called alphabets, $\delta : Q \times \Sigma \rightarrow Q$ = a transition function, q_o is an initial or start state ($q_0 \in Q$), and $F \subseteq S$ is a set of

accepting states. In a non-deterministic automaton, it is possible to have more than one edge with the same symbol. Thus to transit from one state to the other, the edge must be labeled with an input symbol. However, where there is no input symbol, the edge is marked with an empty string (λ) also or the epsilon (ε) are also used in NFA [15].

In order to determine if an input symbol is set of strings, it is possible to use a finite automaton. This is done by selecting a particular state of the automaton as the initial or starting state. From the start state, there can be transition without input strings meaning the transition follows an empty string (or epsilon) transition to the next state or a symbol can be labeled by that input character. Thus the resultant automaton is nondeterministic since the transition is uniquely determined by the current state and the input symbol. This can be due to the fact that certain actions led to accepting whereas others do not. Thus, it suffixes to say that the current state is indicated as the accepted state if all the input symbols are read. In this case, the input symbol is contained in the language defined by the automaton. Therefore, nondeterministic finite automata can formally be defined as follows:

Definition 2 (NFA): *A nondeterministic finite automata (NFA) is a quintuple $N = (Q, \Sigma, \delta, q_o, F)$, where Q is a finite set of states, Σ is of input symbols called alphabets, $\delta : Q \times \Sigma \rightarrow Q$ = a transition function, q_o is an initial or start state ($q_0 \in Q$), and $F \subseteq S$ is a set of accepting states.*

2.2 Deterministic finite automata (DFA)

A deterministic finite automaton (DFA) is a computational device that processes strings or input symbols of some alphabet(s) or input strings and produces a result that is either accepted or rejected [16,17]. If the string is accepted, it means that the string is part of a recognized language otherwise; it is not [18]. Thus if the language is accepted, it is then defined as all strings that bring the DFA into an accepting state. As an example, it is possible to construct a DFA that accept a language, a^n, where $n > 2$ as follows.

Definition 3 (DFA): *A deterministic finite automaton (DFA), $\mathcal{D}$, is a quintuple $\mathcal{D} = (Q, \Sigma, \delta, q_o, F)$, where Q is a finite set of states, Σ is of input symbols called alphabets, $\delta : Q \times \Sigma \rightarrow Q$ = a transition function, q_o is an initial or start state ($q_0 \in Q$), and $F \subseteq S$ = a set of accepting states.*

DFAs are usually represented by transition diagrams with the states represented as circles starting from the initial state to all the other states apart from the final or accepting state which is represented by double circles. If the language is recognized, then it will end in the final or accepting state; otherwise, the language is rejected. The states are connected by arrows indicating the direction in which the transition should move next [19,20]. However, there are some limitations to the kind of language the DFA can recognize. The number of input symbols and the number of states in the DFA must be finite for the DFA to be recognized.

3 Equivalent Automata

Two automata are equivalent if both they produce identical output for each input string [21]. This way, in can be said that two finite automata are equivalent if they both accept the same language [22,23].

Definition 4 (Equivalent automata)*: Let Q and Q^l be two finite-state automata with the same set of input symbols I. Let $L(Q)$ denote the language accepted by Q and $L(Q^l)$. We say Q is equivalent to Q^l if and only if $L(Q) \equiv L(Q^l)$.*

Definition 5 (Equivalence of finite state automata)*: Two automata A_1 and A_2 are equivalent ($A_1 \equiv A_2$) if they both have the same input and output symbols and produce the same output for any input string if they both start from their respective initial states.*

***Definition 6 (Equivalence of states)**: Two finite-states automata A_1 and A_2 (which may or may not be the same as A_1) are equivalent ($A_1 \equiv A_2$) if t they both have the same input and output strings and if for any input string, automaton A_1 started from state q_1 produces the same output as automaton A_2 which started from state q_2.*

***Definition 7 (Strongly connected automata)**: A finite-state automaton A is strongly connected if, for any two states s_1 and s_2 in the state space of A, there is a finite sequence of transitions that transit A from state s_1 to state s_2.*

In order to determine if two automata are equivalent, output devices are usually attached to the states of a finite automata to show if they are accepting or rejecting states. As an example, suppose there is a finite automaton with accepting states that produces an output of 1 and rejecting states with an output 0, then such a finite automaton can be regarded as an input/output device whose values consists of strings and output values consists of strings of 0's and 1's. Therefore, a finite automaton is a machine that processes input strings and produces outputs strings whether these output strings are accepted or not. Fig. 1 shows an algorithm that can be used to determine if two automata are equivalent.

Algorithm 1: Algorithm for Two Equivalent Automata

Input: Two different deterministic finite automata (DFA)
Output: Two automata that are equivalent
begin
for automata A determine the:
0-equivalence classes;
1-equivalent classes;
2-equivalent classes **;**
construct the automata
end // end for
begin
for automata A^1 determine the:
0-equivalence classes;
1-equivalent classes ;
2-equivalent classes;
construct the automata
end; //end for
if transition diagrams A and A^1 are identical **then**
automata $A \equiv A^1$
if A is equivalent to A^1 **then**
$A \equiv A^1$
endif; // end if
endif; //end if
end.

Fig. 1. Determiningr two equivalent automata

4 Examples of Equivalent Automata

In this section, we solve some problems and show by proving that the set of automata in consideration are equivalent or not. We also supported our claims with reason why such automata are considered equivalent.

Theorem 1: Prove that the two automata in A and A^1 in Fig. 2 are equivalent.

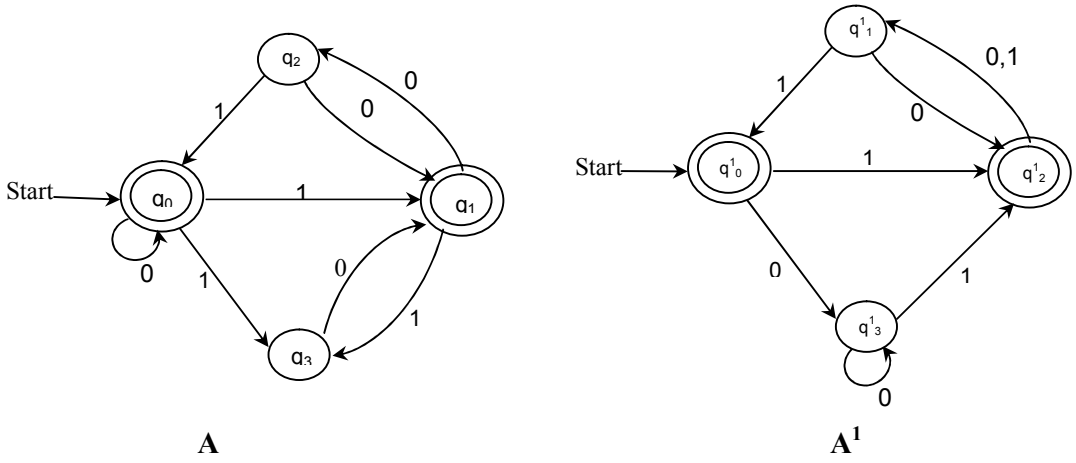


Fig. 2. Two automata A and A^1

Proof

In order to show that the two automata are equivalent, we first consider automaton A in Fig. 2.

For automaton A: the 0-equivalence classes are determined by first identifying the accepting and rejecting state. In the case of automaton A, the states that are accepting and rejecting are:

- q_0 and q_1 are accepting states
- q_2 and q_3 are not accepting states

We grouped the accepting states as a set and the non-accepting states as a set. That is, $\{q_0, q_1\}$ and $\{q_2, q_3\}$ respectively. Next, we consider the 1-equivalence classes.

The 1-equivalence classes are as follows:

Since, $N(q_0, 1)$ = but $N(q_1, 1) = q_3$, then we choose $\{q_0\}$, $\{q_1\}$, and $\{q_2, q_3\}$ as 1-equivalent. Thus the pair of states below shows whether they are 0 or 1-euqivalence or both q_0 and q_1 are not 1-equivalent

- q_1 is not 0-equivalent
- q_2 and q_3 are 1-equivalent

The 2-equivalence classes are as follows:

Since q_2 and q_3 are 1-equivalent, therefore, we choose states $\{q_0\}$, $\{q_1\}$, and $\{q_2, q_3\}$. From the foregoing, it can be seen that the set of 1-equivalent classes are the same as the set of 2-equivalent classes. Based on the *-equivalent classes, it follows that the *-equivalent classes are:

$\{q_0\}$, $\{q_1\}$, and $\{q_2, q_3\}$

which state that if A is a finite-state automaton, then for some integer $i \geq 0$, the set of i-equivalent classes are of states A equals the set of (i + 1)-equivalent classes of states of A, and for all such i these are both equal to the set of *-equivalent classes of states of A.

From the automaton A^1: Based on similar reasons given for automaton A, we determine the 0-equivalent classes.

The 0-equivalence classes are:

$\{q^1_0, q^1_2, q^1_3\}$ and $\{q^1_1\}$

The 1-equivalence classes are:

$\{q^1_0, q^1_3\}$, $\{q^1_2\}$, and $\{q^1_1\}$

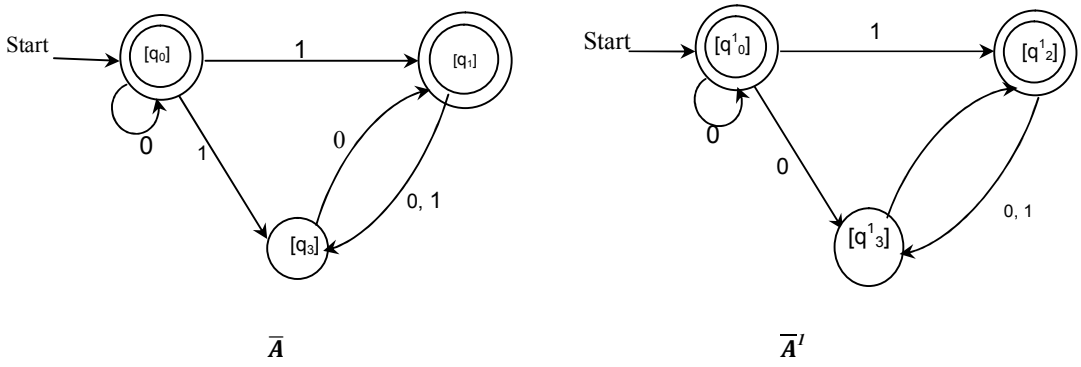
The 2-equivalence classes are:

Since q_2 and q_3 are 1-equivalent, therefore, we choose states $\{q^1_0]$, q^1_1, $\{q^1_2, q^1_3\}$

Therefore, it can be seen that the set of 1-equivalent classes are the same as the set of 2-equivalent classes. Based on the *-equivalent classes, it follows that the *-equivalent classes are:

$\{q^1_0, q^1_3\}$, $\{q^1_2\}$, and $\{q^1_1\}$

Based on the proof above, it can be seen that the two automata A and A^1, are equivalent. Finally, the resulting equivalent automata for automata A and A^1 in Fig. 3 are:

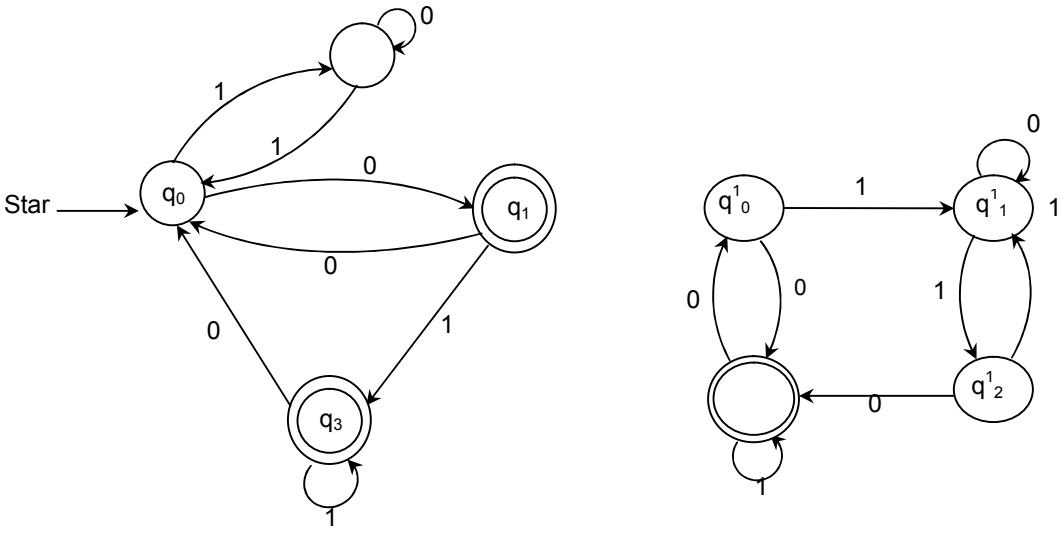


$\overline{A}$ $\overline{A}^I$

Fig. 3. Finite automata with minimized states equivalent to A and A^1

These two automata are equivalent except that the position of state q_1 and q_2 changes.

Theorem 2: Prove that the two automata B and B^1 in Fig. 4 are equivalent.



B B[1]

Fig. 4. Two automata B and B[1]

Proof

We first consider automaton B.

For automaton B: the 0-equivalence classes are:

First, we identify the accepting and non-accepting state.

- q_0 and q_1 are accepting states
- q_2 and q_3 are not accepting states

We then group the accepting states as a set and the non-accepting states are a set.

Next, we consider the 1-equivalence classes

The 1-equivalence classes are as follows:

Since $N(q_0, 1)$ = but $N(q_1, 1) = q_3\{q_0\}$, $\{s_1\}$, and $\{q_2, q_3\}$, then q_0 and q_1 are not 1-equivalent; q_1 is not 0-equivalent; and q_2 and q_3 are 1-equivalent

The 2-equivalence classes are as follows:

Also, since q_2 and q_3 are 1-equivalent; $\{q_0\}$, $\{q_1\}$, and $\{q_2, q_3\}$

From the foregoing, it can be seen that the set of 1-equivalence classes are the same as the set of 2-equivalence classes. Based on the *-equivalence classes, it follows that the *-equivalence classes are:

$\{q_0\}$, $\{q_1\}$, and $\{q_2, q_3\}$

which state that if A is a finite-state automaton, then for some integer $i \geq 0$, the set of i-equivalence classes are of states A equals the set of (i + 1)-equivalent classes of states of X, and for all such i these are both equal to the set of *-equivalent classes of states of A.

Automaton B^1: Based on similar reasons given for automaton B, we determine the 0-equivalence classes.

The 0-equivalence classes are:

$\{q^1_0, q^1_2, q^1_3\}$ and $\{q^1_1\}$

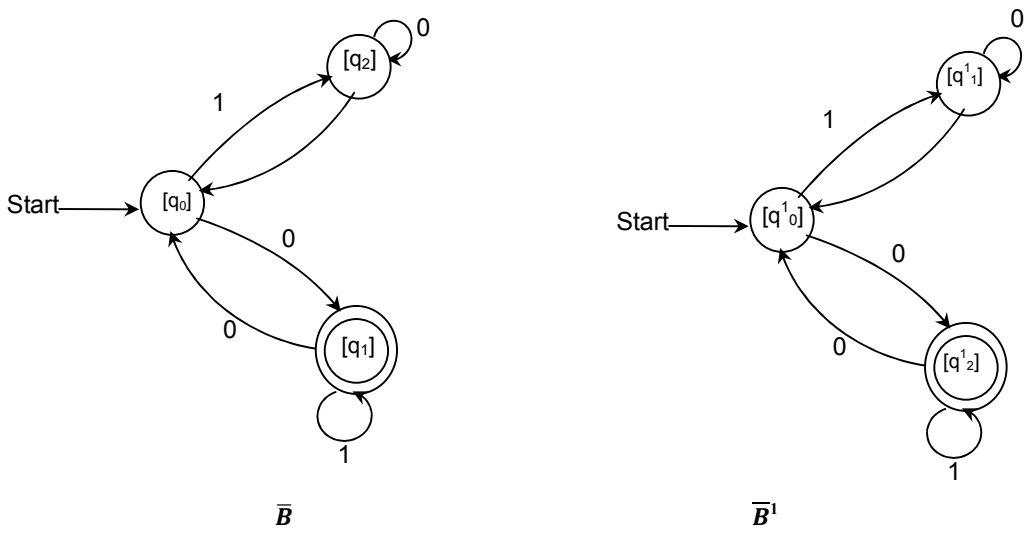
The 1-equivalence classes are:

$\{q^1_0, q^1_3\}$, $\{q^1_2\}$, and $\{q^1_1\}$

The 2-equivalence classes are the same just as in automaton B, which makes them the same as *-equivalence classes. Therefore, the *-equivalence classes are:

$\{q^1_0, q^1_3\}$, $\{q^1_2\}$, and $\{q^1_1\}$

Based on the proof above, it can be seen from Fig. 5 ($\overline{\boldsymbol{B}}$ and $\overline{\boldsymbol{B}}\mathbf{1}$) that automata B and B^1, are equivalent.



$\overline{\boldsymbol{B}}$ $\overline{\boldsymbol{B}}^1$

Fig. 5. Finite automata with minimized states equivalent to B and B^1

Also, these two automata are equivalent except for the position of states q_1 and q_2.

5 Conclusion

Often times, two automata that are equivalent could be an NFA that is converted to DFA or a DFA that is minimized into an equivalent DFA. A nondeterministic finite automaton can be converted to a deterministic finite automaton having fewer states when compared with the original NFA. The two finite-state automata are equivalent if they have the same power and can therefore be used to compute the same regular expression. In this paper, we develop an algorithm that can be used to simulate two equivalent finite state automata. Two finite-state automata are said to be equivalent if they produce identical output for each input string. The concept of finite automaton has been in existence for decades. It is very useful when developing compilers in programming languages and programming in general. It captures all possible states and

transitions that a machine can take while reading a sequence of input strings or symbols. Examples of the use of the algorithm are provided and their Results show that they are equivalent in all respects.

Competing Interests

Author has declared that no competing interests exist.

References

[1] Zhang J, Qian Z. The equivalent conversion between regular grammar and finite automata. Journal of Software Engineering and Applications. 2013;6:33-37.

[2] Pandey H, Singh VK, Pandey A. A new NFA reduction algorithm for state minimization problem. Int'l Journal of Applied Information System (IJAIS). 2015;8(3):27-30.

[3] Aho AV, Sethi H, Ullman JD. Compilers: Principles, techniques and tools. Addison-Wesley Publisher Co., Reading, Massachusetts; 1988.

[4] Freund Y, Kearns M, Ron M, Rubinfeld R, Schapire RE, Sellie L. Efficient learning of typical finite automata from random walks. 25th ACM Symposium on Theory of Computing. 1997;315-324.

[5] Ogheneovo EE. An alternative algorithm for simulating finite-state automata. The Journal of Digital Innovations & Contemporary Research in Science and Engineering. 2016;4(4):71-82.

[6] Geffert V, Guillon B, Pighizzini G. Two-way automata: Making choices only at the Endmakers. In Dediu, A. H. and Martin-Vide, C. (Ed.): LATA, Lecture Notes in Computer Science 7183, Springer. 2012;264-276.
DOI: 10.1007/978-3-642-28332-1.23

[7] Pandey H, Singh VK, Pandey A. A new NFA reduction algorithm for state minimization problem. Int'l Journal of Applied Information System (IJAIS). 2015;8(3):27-30.

[8] Rastogi D, Singh R. Technique to remove indistinguishable state with unreachable state and dead state from deterministic finite automata. Int'l Journal f Computer Application. 2014;94(1):35-39.

[9] Hopcroft JE, Ullman JD. Introduction to automata theory, languages and computation. Addison-Wesley, Readings, Massachusetts, USA; 1979.

[10] Mera F, Pighizzini G. Complimenting unary nondeterministic automata (2004). Theoretical Computer Science. 2004;330:349-360.

[11] Hozler M, Kutrib M. Descriptional and computational complexity of finite automata –A survey (2011). Information and Computation. 2011;209:456-470.

[12] Rivest RL, Schapire RE. Diversity-based inference of finite automata. Journal of the ACM. 1994;41(3):555-589.

[13] Jirásková G. Note on minimal finite automata. In Gigall, J. and Kolman, P. (Eds.): MFCS 2001, LNCS 2136, Springer-Verlag, Berlin, Heidelberg. 2001;421-431.

[14] Pighizzini G. Two-way finite automata: Old and recent results. In E. Formenti (Ed.): AUTOMATA and JAC 2012 Conferences, EPTCS. 2012;90:3-20.
DOI: 10.4204/EPTCS.90.1

[15] Plump D, Suri R, Singh A. Minimizing finite automata with graph programs. Electronic Communications of the EASST. 2011;39:1-15.

[16] Gruber H, Holzer M. From finite automata to regular expression and back – A summary on descriptional complexity. In Esik, Z. and Fulop, Z. (Eds.): Automata and Formal Languages. 2014 (AFL 2014), EPTCS. 2014;151:25-48.
DOI: 10.4204/EPTCS.1512

[17] Jirásková G, Palmovsky M. In processing of CEUR'13. 2013;1003:94-100.

[18] Almeida M, Moreira N, Reis R. Testing the equivalence of regular langages. In Dassow, J. Pighizzini, G. and Truthe, B. (Eds.): 11th Int'l Workshop on Descriptional Complexity of Formal Systems (DCFs'09), EPTCS. 2009;3:47-57.
DOI: 10.4204/EPTCS.3.4

[19] Axdsen HB, Holzer M, Kutrib M. Springer Int'l Publishing, Switzerland. In Han, Y.-S., Salomaa, K. (Eds.): CIAA 2016, LNCS 9705. 2016;15-26.
DOI: 10.1007/97S-3-319-40946-7_2

[20] Brüggemann-Klein A. Regular expressions into finite automata. Theoretical Computer Science. 1993; 120:197-213.

[21] Tewari A, Srivastava U, Gupta P. Parallel DFA Minimization Algorithm. In Sahni et al. (Eds.) HiPC 2002, LNCS 2552. Springer-Verlag, Berlin Heidelberg. 2002;34-40.

[22] Vázquez de Parge M, García P, López D. A polynomial double reversal minimization algorithm for deterministic finite automata. Theoretical Computer Science. 2013;487:17-22.

[23] Wieczorek W. Induction of non-deterministic finite automata on supercomputers. In Heinz, J. de la Higuera, C. and Oates, T. JMLR: In Proceedings of the 11th ICGI International Workshop and Conference. 2012;2:237-242.

Peer-review history:
The peer review history for this paper can be accessed here (Please copy paste the total link in your browser address bar)
http://www.sciencedomain.org/review-history/27526