International Journal of Advanced Science and Technology
Vol.118 (2018), pp.151-160
http://dx.doi.org/10.14257/ijast.2018.118.14

Analysis of Regular Operations Application and Finite Automa

Daisy T. Endencio-Robles[1*] and Rosslin John Robles[1]

[1] *University of San Agustin, Philippines*
dendencio@usa.edu.ph, rjrobles@usa.edu.ph

Abstract

Arithmetic operations create a new value from 1 or 2 existing values (eg 2 + 3). Regular operations create a new language from 1 or 2 existing languages. In this paper, we reviewed the state complexities of some basic operations on regular languages, The state complexity of reversals of regular languages, State complexity of some operations on binary regular languages and the state complexity of basic operations on suffix-free regular languages. We also reviewed the operations on finite automata.

Keywords: *regular operations, regular languages, state complexity*

1. Introduction

Given some languages, what new languages can we construct from these languages which are regular languages (can be generated by a deterministic finite automaton *(DFA)*? The following are a list of definitions for some regular operations. These are not all regular operations, by the way, but an important class of such operations. Let *A* and *B* be languages (sets of strings.) Then we define the union of *A* and *B* as the new language [1]

$$A \bigcup B = \{w | w \epsilon A or w \epsilon B\}$$

Thus the union of *A* and *B* is a new language which includes all of the strings which are in A and all of the strings which are in *B*. Similarly, we can define the intersection of *A* and *B* as the new language

$$A \bigcap B = \{w | w \epsilon A and w \epsilon B\}$$

Thus the intersection of *A* and *B* is a new language which includes all of the strings that are in both *A* and *B*. We define the concatenation of *A* and *B* as the new language. [1]

$$AoB = \{wv | w \epsilon A and v \epsilon B\}$$

In other words, the concatenation of the two languages *A and B* is made up of strings which come first from *w* and then from *v*. Note that *A* o *B* is not equal to *B* o *A* (the o operator is not commutative.) [1]

$$A* = \{x_1 x_2 .... x_k \geq 0 \text{ and each } x_1 \epsilon A\}$$

We define the star of a language *A* as the new language.

Received (May 13, 2018), Review Result (August 7, 2018), Accepted (August 13, 2018)
* Corresponding Author

ISSN: 2005-4238 IJAST
Copyright ⓒ 2018 SERSC Australia

2. Regular Operation Application

2.1. The State Complexities of some Basic Operations on Regular Languages [2]

2.1.1. State Complexity of Concatenation of Two Regular Languages. A general example which shows that for any $m >=1$ and $n > 1$ there exist an m-state DFA A and an n-state DFA B such that any DFA accepting $L(A)L(B)$ needs at least $m2n-2n-1$ states. Then we show that for any pair of complete m-state DFA A and n-state DFA B defined on the same alphabet, there exists a DFA with at most $m2n-2n-1$ states which accepts $L(A)L(B)$. In the case of $n = 1$ and $m >= 1$, we show that m states are sufficient and necessary in the worst case for a DFA to accept $L(A)L(B)$. [2]

2.1.2. State Complexity of Star Operation on Regular Languages. In [3], an example is given to show that any DFA accepting the star of an n-state DFA language needs at least $2n-1$ states in some cases for $n > 0$. Here we improve that result and show that $2n-1+2n-2$ is necessary in the worst case for a DFA to accept the star of an n-state DFA language for each $n > 1$. We use a very different technique and use a two-letter alphabet. However, we give the sufficient condition first. [2]

2.2. The State Complexity of Reversals of Regular Languages

If the state complexity of a regular language L equals n, then the state complexity of its mirror image $mi(L)$ is at most $2n$. In [4], various classes of languages for which this maximal blow-up actually occurs were presented. We have also considered cases where this phenomenon never occurs. However, a necessary and sufficient condition for this phenomenon to occur is still missing. We have also considered the special case of finite languages and exhibited a variety of growth types in the state complexity when a finite language is replaced by its mirror image.

Since the mirror image of a regular language L with state complexity n is always accepted by a nondeterministic finite automaton with n states, our results can also be viewed as a contribution to the trade-off between nondeterminism and determinism.

A related topic about this trade-off concerns languages over one letter. It is well known that, in this case, the deterministic state complexity is not always polynomial in terms of the nondeterministic one, but no explicit bounds have been given.

2.3. State Complexity of some Operations on Binary Regular Languages

In [5], the author considered the concatenation of languages represented by eterministic finite automata, and the reversal of languages represented by nondeterministic finite automata.

2.3.1. Catenation Operation. The state complexity of the catenation of regular languages represented by deterministic finite automata was studied by Yu et al. [6]. They showed that $2^n - 2^{n-1}$ states are sufficient for a DFA to accept the catenation of an m-state DFA language and an n-state DFA language. In the case of $n = 1$, the upper bound m was shown to be tight for a unary alphabet. In the case of $m = 1$ and $n>=2$, the worst case $m2n - 2^{n-1}$ was given by the catenation of two binary languages. Otherwise the upper bound $m2^n - 2^{n-1}$ was proved to be tight for a three-letter alphabet. The next theorem shows that the upper bound can be reached by the catenation of two binary languages.

Theorem 1. For any integers $m>=2$ and $n>=2$, there exist a binary DFA A of m-states and a binary DFA B of n-states such that any DFA accepting the language $L(A)L(B)$ needs at least $m2^n - 2^{n-1}$ states.

 Copyright ⓒ 2018 SERSC Australia

2.3.2. Reversal Operation. It is known that the reversal of any n-state DFA language can be accepted by a DFA of 2^n states and the worst case can be reached by the reversal of a binary DFA language [7]. The upper bound on the size of an NFA accepting the reversal of an n-state NFA language is known to be $n + 1$ and Holzer and Kutrib [8] proved that the upper bound is tight for a three-letter alphabet. The next theorem shows that the upper bound $n + 1$ can be reached by the reversal of a binary n-state NFA language. To obtain the result we use a counting argument. Since the reversal of any 1-state NFA language is the same language, we assume that $>=2$.

Theorem 2. For any integer $n>=2$, there exists a binary NFA D of n states such that anyNFA accepting the reversal of the language L(D) needs at least $n + 1$ states. [5] proved that the upper bounds on the state complexity of these operations, which were known to be tight for larger alphabets, are tight also for binary alphabets.

3. State Complexity of Basic Operations on Suffix-Free Regular Operations

The state complexity of an operation for regular languages is the number of states that are necessary and sufficient in the worst-case for the minimal deterministic finite-state automaton that accepts the language obtained from the operation. [9] established the precise state complexity of catenation, Kleene star, reversal and the Boolean operations for suffix free regular languages.

3.1. Kleene Star and Reversal. Before examining the state complexity of various operations, we establish that any suffix-free (complete) minimal DFA must always have a sink state. Recall that the state complexity of a regular language L is the number of states in its minimal DFA. If L is a regular language, its minimal DFA does not necessarily have a sink state. However, if L is prefix-free, then its minimal DFA A must have a sink state since A is non-exiting. Therefore, we have to verify the existence of the sink state in a suffix-free minimal DFA before investigating the state complexity for each operation. This is crucial for computing the correct state complexity.

4. Finite Automata

Finite automata are a fundamental model of computation that has been systematically studied since the 1950's. A deterministic finite automata (DFA) is a finite-state machine that accepts or rejects strings of symbols and only produces a unique computation of the automaton for each input string.[10] Deterministic refers to the uniqueness of the computation. In search of the simplest models to capture finite-state machines, Warren McCulloch and Walter Pitts were among the first researchers to introduce a concept similar to finite automata in 1943 [11][12].

Copyright ⓒ 2018 SERSC Australia

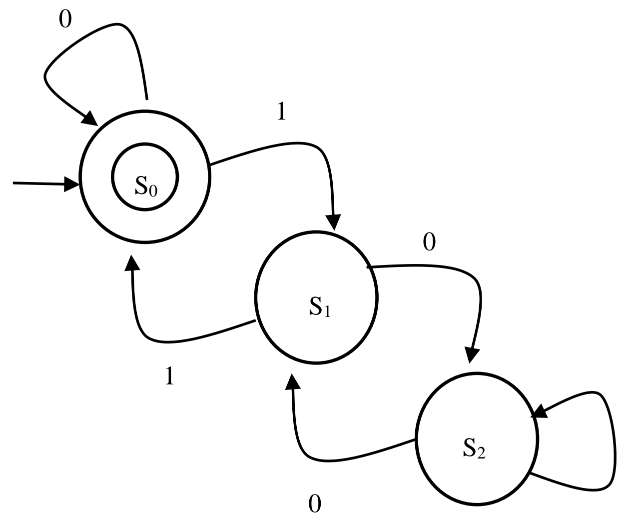

Figure 1. An Example of a Deterministic Finite Automaton

The figure above is an example of a deterministic finite automaton that accepts only binary numbers that are multiples of 3. The state S0 is both the start state and an accept state. In the automaton, there are three states: S0, S1, and S2 (denoted graphically by circles). The automaton takes a finite sequence of 0s and 1s as input. For each state, there is a transition arrow leading out to a next state for both 0 and 1. Upon reading a symbol, a DFA jumps deterministically from one state to another by following the transition arrow. For example, if the automaton is currently in state S0 and the current input symbol is 1, then it deterministically jumps to state S1. A DFA has a start state (denoted graphically by an arrow coming in from nowhere) where computations begin, and a set of accept states (denoted graphically by a double circle) which help define when a computation is successful.

A DFA is defined as an abstract mathematical concept, but is often implemented in hardware and software for solving various specific problems. For example, a DFA can model software that decides whether or not online user input such as email addresses are valid. [13]

DFAs recognize exactly the set of regular languages, [10] which are, among other things, useful for doing lexical analysis and pattern matching. DFAs can be built from nondeterministic finite automata (NFAs) using the powerset construction method.

4.1 Generate and Accept Modes. A DFA representing a regular language can be used either in an accepting mode to validate that an input string is part of the language, or in a generating mode to generate a list of all the strings in the language [14].

The generating mode is similar except that rather than validating an input string its goal is to produce a list of all the strings in the language. Instead of following a single transition out of each state, it follows all of them. In practice this can be accomplished by massive parallelism (having the program branch into two or more processes each time it is faced with a decision) or through recursion. As before, the computation begins at the start state and then proceeds to follow each available transition, keeping track of which branches it took. Every time the automaton finds itself in an accept state it knows that the sequence of branches it took forms a valid string in the language and it adds that string to the list that it is generating. If the language this automaton describes is infinite (*i.e.*, contains an infinite number or strings, such as "all the binary string with an even number of 0s) then the computation will never halt. Given that regular languages are, in general, infinite, automata in the generating mode tends to be more of a theoretical construct. [14]

Copyright ⓒ 2018 SERSC Australia

In the accept mode an input string is provided which the automaton can read in left to right, one symbol at a time. The computation begins at the start state and proceeds by reading the first symbol from the input string and following the state transition corresponding to that symbol. The system continues reading symbols and following transitions until there are no more symbols in the input, which marks the end of the computation. If after all input symbols have been processed the system is in an accept state then we know that the input string was indeed part of the language, and it is said to be accepted, otherwise it is not part of the language and it is not accepted [14].

4.2. DFA as a Transition Monoid. Alternatively, a run can be seen as a sequence of compositions of transition function with itself. Given an input symbol $a \in \sum$, one may write the transition function as, using the simple trick of currying, that is, writing $\delta(q,a) = \delta_a(q)$ for all $q \in Q$. This way, the transition function can be seen in simpler terms: it's just something that δ_a "acts" on a state in Q, yielding another state. One may then consider the result of function composition repeatedly applied to the various functions δ_a, δ_b, and so on. Using this notion we define. Given a pair of letters $a,b \in \sum$,, one may define a new function $\delta_{ab} = \delta_{a+}\delta_b$, by insisting that, where denotes function composition. Clearly, this process can be recursively continued. So, we have following recursive definition $\delta: Q \times \sum^* \to Q$:

$\boldsymbol{\delta (q, \epsilon) = q}$ where ε is empty string and

$\boldsymbol{\delta (q, wa) = \delta_a(\delta (q, w))}$ where $w \in \sum^*$, $a \in \sum$ and $q \in Q$.

$\boldsymbol{\delta}$ is defined for all words $w \in \sum^*$. Repeated function composition forms a monoid. For the transition functions $\boldsymbol{\delta}$, this monoid is known as the transition monoid, or sometimes the transformation semigroup. The construction can also be reversed: given a $\boldsymbol{\delta}$, one can reconstruct a δ, and so the two descriptions are equivalent. [14]

4.3. Closure Properties. If DFAs recognize the languages that are obtained by applying an operation on the DFA recognizable languages then DFAs are said to be closed under the operation. The DFAs are closed under the following operations.

- Union
- Intersection
- Concatenation
- Negation
- Kleene closure
- Reversal
- Init
- Quotient
- Substitution
- Homomorphism

For each operation, an optimal construction with respect to the number of states has been determined in the state complexity research. Since DFAs are equivalent to nondeterministic finite automata (NFA), these closures may also be proved using closure properties of NFA.

Copyright ⓒ 2018 SERSC Australia

4.4. Local Automata. A local automaton is a DFA for which all edges with the same label lead to a single vertex. Local automata accept the class of local languages, those for which membership of a word in the language is determined by a "sliding window" of length two on the word [15, 16].

A Myhill graph over an alphabet A is a directed graph with vertex set A and subsets of vertices labelled "start" and "finish". The language accepted by a Myhill graph is the set of directed paths from a start vertex to a finish vertex: the graph thus acts as an automaton.[15] The class of languages accepted by Myhill graphs is the class of local languages [15].

4.5. Random. When the start state and accept states are ignored, a DFA of n states and an alphabet of size k can be seen as a digraph of n vertices in which all vertices have k out-arcs labeled $1, \cdots, k$ (a k-out digraph). It is known that when $k \geq 2$ is a fixed integer, with high probability, the largest strongly connected component (SCC) in such a k-out digraph chosen uniformly at random is of linear size and it can be reached by all vertices. [8] It has also been proven that if k is allowed to increase as n increases, then the whole digraph has a phase transition for strong connectivity similar to Erdős–Rényi model for connectivity [18].

In a random DFA, the maximum number of vertices reachable from one vertex is very close to the number of vertices in the largest SCC with high probability [17, 19]. This is also true for the largest induced sub-digraph of minimum in-degree one, which can be seen as a directed version of 1-core [18].

4.6. Pros and Cons of deterministic finite-state Automata. DFAs are one of the most practical models of computation, since there is a trivial linear time, constant-space, online algorithm to simulate a DFA on a stream of input. Also, there are efficient algorithms to find a DFA recognizing:

- the complement of the language recognized by a given DFA.
- the union/intersection of the languages recognized by two given DFAs.

Because DFAs can be reduced to a canonical form (minimal DFAs), there are also efficient algorithms to determine:

- whether a DFA accepts any strings
- whether a DFA accepts all strings
- whether two DFAs recognize the same language
- the DFA with a minimum number of states for a particular regular language

DFAs are equivalent in computing power to nondeterministic finite automata (NFAs). This is because, firstly any DFA is also an NFA, so an NFA can do what a DFA can do. Also, given an NFA, using the powerset construction one can build a DFA that recognizes the same language as the NFA, although the DFA could have exponentially larger number of states than the NFA. [14]

On the other hand, finite state automata are of strictly limited power in the languages they can recognize; many simple languages, including any problem that requires more than constant space to solve, cannot be recognized by a DFA. The classical example of a simply described language that no DFA can recognize is bracket language, *i.e.*, language that consists of properly paired brackets such as word "(()())". No DFA can recognize the bracket language because there is no limit to recursion, *i.e.*, one can always embed

Copyright ⓒ 2018 SERSC Australia

another pair of brackets inside. It would require an infinite amount of states to recognize. Another simpler example is the language consisting of strings of the form a^nb^n—some finite number of a's, followed by an equal number of b's. [14]

4.7. Ambiguity, Nondeterminism and State Complexity. Ambiguity is a fundamental concept in grammar derivations. A regular expression is unambiguous if it denotes each string in at most one way. A nondeterministic finite automaton (NFA) is unambiguous if each string has at most one accepting computation. The degree of ambiguity of an NFA A on a string $\boldsymbol{w}$ is the number of accepting computations of A on $\boldsymbol{w}$. The degree of ambiguity of A is the maximal degree of ambiguity of A on any input string, if the maximum exists, and in this case A is said to be finitely ambiguous. Otherwise the degree of ambiguity of A can be measured as a function of the length of the inputs [20].

4.7.1. The degree of ambiguity is defined in terms of the number of accepting computations, and does not directly limit the amount of nondeterminism, or the amount of guessing, used by an automaton. In an unambiguous NFA, even though an accepting computation is unique, the computation may include any number of nondeterministic steps - unambiguity implies just that at any nondeterministic step at most one choice can lead to acceptance. In order to develop a quantitative understanding of the power of nondeterminism, one can directly measure the number of nondeterministic steps used by an NFA. [20]

An ambiguous grammar is a context-free grammar for which there exists a string that can have more than one leftmost derivation or parse tree, while an unambiguous grammar is a context-free grammar for which every valid string has a unique leftmost derivation or parse tree. Many languages admit both ambiguous and unambiguous grammars, while some languages admit only ambiguous grammars. Any non-empty language admits an ambiguous grammar by taking an unambiguous grammar and introducing a duplicate rule or synonym (the only language without ambiguous grammars is the empty language). A language that only admits ambiguous grammars is called an inherently ambiguous language, and there are inherently ambiguous context-free languages. Deterministic context-free grammars are always unambiguous, and are an important subclass of unambiguous grammars; there are non-deterministic unambiguous grammars, however.

For computer programming languages, the reference grammar is often ambiguous, due to issues such as the dangling else problem. If present, these ambiguities are generally resolved by adding precedence rules or other context-sensitive parsing rules, so the overall phrase grammar is unambiguous. The set of all parse trees for an ambiguous sentence is called a parse forest [21].

4.7.1.1. Trivial Language. The simplest example is the following ambiguous grammar for the trivial language, which consists of only the empty string:

$$A \rightarrow A \mid \varepsilon$$

meaning that a production can either be itself again, or the empty string. Thus the empty string has leftmost derivations of length 1, 2, 3, and indeed of any length, depending on how many times the rule $A \rightarrow A$ is used.

This language also has the unambiguous grammar, consisting of a single production rule:

$$A \rightarrow \varepsilon$$

meaning that the unique production can only produce the empty string, which is the unique string in the language.

Copyright ⓒ 2018 SERSC Australia

In the same way, any grammar for a non-empty language can be made ambiguous by adding duplicates.

4.7.1.2. Unary String. The regular language of unary strings of a given character, say 'a' (the regular expression a*), has the unambiguous grammar:

$A \rightarrow aA \mid \varepsilon$

…but also has the ambiguous grammar:

$A \rightarrow aA \mid Aa \mid \varepsilon$

These correspond to producing a right-associative tree (for the unambiguous grammar) or allowing both left- and right- association. This is elaborated below.

4.7.2. A nondeterministic finite automaton (NFA), or nondeterministic finite state machine, does not need to obey these restrictions. In particular, every DFA is also an NFA. Sometimes the term NFA is used in a narrower sense, referring to a NFA that is not a DFA. An NFA, similar to a DFA, consumes a string of input symbols. For each input symbol, it transitions to a new state until all input symbols have been consumed. Unlike a DFA, it is non-deterministic, i.e., for some state and input symbol, the next state may be nothing or one or two or more possible states. Thus, in the formal definition, the next state is an element of the power set of the states, which is a set of states to be considered at once. The notion of accepting an input is similar to that for the DFA. When the last input symbol is consumed, the NFA accepts if and only if there is some set of transitions that will take it to an accepting state. Equivalently, it rejects, if, no matter what transitions are applied, it would not end in an accepting state.

4.7.2.1. Equivalence to DFA. A Deterministic finite automaton (DFA) can be seen as a special kind of NFA, in which for each state and alphabet, the transition function has exactly one state. Thus, it is clear that every formal language that can be recognized by a DFA can be recognized by a NFA.

Conversely, for each NFA, there is a DFA such that it recognizes the same formal language. The DFA can be constructed using the powerset construction.

This result shows that NFAs, despite their additional flexibility, are unable to recognize languages that cannot be recognized by some DFA. It is also important in practice for converting easier-to-construct NFAs into more efficiently executable DFAs. However, if the NFA has n states, the resulting DFA may have up to 2n states, which sometimes makes the construction impractical for large NFAs.

4.7.2.2. NFA with ε-moves. Nondeterministic finite automaton with ε-moves (NFA-ε) is a further generalization to NFA. This automaton replaces the transition function with the one that allows the empty string ε as a possible input. The transitions without consuming an input symbol are called ε-transitions. In the state diagrams, they are usually labeled with the Greek letter ε. ε-transitions provide a convenient way of modeling the systems whose current states are not precisely known: i.e., if we are modeling a system and it is not clear whether the current state (after processing some input string) should be q or q', then we can add an ε-transition between these two states, thus putting the automaton in both states simultaneously.

 Copyright ⓒ 2018 SERSC Australia

5. Conclusion

In conclusion, the state complexities of some basic operations on regular languages shows that the result of $2^{n-1}+2^{n-1}$ states is obtained for the star of an n-state DFA language, $n>1$. If the state complexity of a regular language L equals n, then the state complexity of its mirror image $mi(L)$ is at most $2n$. [2] have presented various classes of languages for which this maximal blow-up actually occurs. [2] have also considered cases where this phenomenon never occurs. [5] have obtained several results concerning the state complexity of some operations on binary regular languages. [5] proved that some upper bounds which were known to be tight for larger alphabets are tight likewise for binary alphabets. [9] have established the tight [5] state complexity bounds for each of the operations using languages over a fixed alphabet. However, the constructions usually require an alphabet of size 3 or 4 and, then, for most operations, it is open whether or not the upper bound for the state complexity of each operation can be reached using a small size alphabet. We also reviewed the Finite automata as fundamental model of computation including modes such as Accept and Generate, DFA as a transition monoid and Closure properties.

References

[1] D. Bacon, "Introduction to Formal Methods in Computer Science Regular Operations on Languages", https://courses.cs.washington.edu/courses/cse322/08au/lec3.pdf Accessed: **(2017)** January.
[2] S. Yu, Q. Zhuang and K. Salomaa, "The state complexities of some basic operations on regular languages", Theoretical Computer Science, Elsevier, vol. 125, **(1994)**, 315328.
[3] B. Ravikumar and O. H. Ibarra, "Relating the type of ambiguity of finite automata to the succinctness of their representation", SIAM J. Comput., vol. 18, no. 6, **(1989)**, 12631282.
[4] A. Salomaa, D. Wood and S. Yu, "On the state complexity of reversals of regular languages", Theoretical Computer Science, Elsevier, vol. 320, **(2004)**, 315329.
[5] G. Jirásková, (2005), "State complexity of some operations on binary regular languages", Theoretical Computer Science, Elsevier, vol. 330, **(2005)**, 287298.
[6] S. Yu, Regular languages, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, vol. 1, Springer, Berlin, New York, pp. 41110 (Chapter 2).
[7] E. Leiss, "Succinct representation of regular languages by boolean automata", Theoret. Comput. Sci., vol. 13, **(1981)**.
[8] M. Holzer, K. Salomaa and S. Yu, "On the state complexity of k-entry deterministic finite automata", J. Automat. Lang. Comb., vol. 6, **(2001)**, 453466.
[9] Y.-S. Han and K. Salomaa, "State complexity of basic operations on suffixfree regular languages", Theoretical Computer Science, Elsevier, vol. 410, **(2009)**, 25372548.
[10] E. Leiss, "Succinct representation of regular languages by boolean automata", Theoret. Comput. Sci., vol. 13, **(1981)**.
[11] J. E. Hopcroft, R. Motwani and J. D. Ullman, "Introduction to Automata Theory, Languages, and Computation (2 ed.)", Addison Wesley. ISBN 0-201-44124-1. Retrieved 19 November 2012, **(2001)**.
[12] W. S. McCulloch and W. Pitts, "A Logical Calculus of the Ideas Immanent in Nervous Activity", Bulletin of Mathematical Biophysics, vol. 5, no. 4, **(1943)**, pp. 115-133.
[13] M. O. Rabin and D. Scott, "Finite automata and their decision problems", IBM J. Res. Develop., **(1959)**, pp. 114-125.
[14] P. Gouda, Prabhakar, Application of Finite automata, Academic Dictionaries and Encyclopedias, "Deterministic finite-state machine", http://enacademic.com/dic.nsf/enwiki/349104.
[15] M. V. Lawson, "Finite automata", Chapman and Hall/CRC. ISBN 1-58488-255-7. Zbl 1086.68074, **(2004)**.
[16] J. Sakarovitch, "Elements of automata theory", Translated from the French by Reuben Thomas. Cambridge: Cambridge University Press. ISBN 978-0-521-84425-3. Zbl 1188.68177, **(2009)**.
[17] A. A. Grusho, "Limit distributions of certain characteristics of random automaton graphs", Mathematical Notes of the Academy of Sciences of the USSR. 4: 633–637. doi:10.1007/BF01095785, **(1973)**.
[18] X. S. Cai and L. Devroye, "The graph structure of a deterministic automaton chosen at random: full version", arXiv:1504.06238
[19] A. Carayol and C. Nicaud, "Distribution of the number of accessible states in a random deterministic automaton", **(2012)**.
[20] Y.-S. Han, A. Saloma and K. Saloma, "Ambiguity, Nondeterminism and State Complexity of Finite Automata", Acta Cybernetica, vol. 23, **(2017)**, pp. 141-157.

Copyright ⓒ 2018 SERSC Australia

[21] M. Tomita, "An efficient augmented-context-free parsing algorithm", Computational linguistics, vol. 13, no. 1-2, **(1987)**, pp. 31-46.

Copyright ⓒ 2018 SERSC Australia