Baltic J. Modern Computing, Vol. 7 (2019), No. 3, pp. 342–353
https://doi.org/10.22364/bjmc.2019.7.3.02

Transition Function Complexity of Finite Automata

Māris VALDATS

University of Latvia, Riga, Raina Boulvd. 19

d20416@lu.lv

Abstract. This paper considers the state assignment problem of finite automata from the complexity point of view. A new complexity measure for finite automata and regular languages, BC-complexity, is introduced, which essentially is the circuit complexity of the transition function of the automaton.
BC-complexity of regular languages is compared with their state complexity and matching upper and lower bounds are obtained for almost all languages with given state complexity. Such bounds are obtained also with a respect to the nondeterministic state complexity but in this case they are not matching. It is proved that the minimization of finite automata can lead to a superpolynomial increase of their BC-complexity.
Finite automata represented with a Boolean circuit are considered also from the computational complexity point of view. It is shown that many simple problems (state reachability or equivalence, minimization of automata) in this representation are PSPACE-complete.

Keywords: finite automata, complexity, circuit representation

1 Introduction

State complexity of finite automata and regular languages has been studied since the beginning of the automata theory and since then it has been the main measure of complexity in automata theory. But, although it works well for automata in the standard representation, sometimes it does not fully reflect the ”intuitive” complexity of the automaton when we come to its implementation.

State assignment problem is a problem of finding an encoding of the states of an automaton that allows a ”simple” implementation of its transition function and it also has been studied since the beginning of automata theory (Hartmanis and Stearns, 1962). This ”simplicity” can be interpreted in various ways. While standard optimization methods try to minimize the dependencies among state variables that leads to an effective implementation of the transition function as a Boolean circuit (Kohavi and Jha, 2009), there are also other approaches that, for example, try to minimize the average switching

of memory elements (Kajstur and Kania, 2017) that corresponds to the minimal power dissipation of the circuit.

Although state assignment problem has been studied for more than 50 years, it has been considered only as a an optimization problem. This can be illustrated by the fact that its goal in many papers is called "minimizing the area of the circuit" (De Micheli et al., 1985), showing the practical nature of this research. But state assignment problem is also a natural complexity problem, that until now has not been considered in the literature as that. The main objective of this work is a theoretical analysis of this model from the descriptional and computational complexity points of view.

When we express a finite automaton as a Boolean circuit then intuitively its number of states does not seem to be the best complexity measure as the complexity of this circuit can vary for automata with the same number of states. Therefore in the first part of this work we introduce a complexity measure for the circuit representation of finite automata, BC-complexity.

BC-complexity is then compared to the state and nondeterministic state complexities of finite automata. In both cases upper and lower bounds are obtained and in the case of the deterministic state complexity they are matching for almost all languages. This is the so called Shannon effect.

It is known since sixties (Hartmanis and Stearns, 1962) that state minimization of a DFA can lead to an increase in the complexity of the circuit describing it. But until now it has been justified only with some small examples. In this work it is proved that state minimization of a DFA can lead to a superpolynomial increase of its BC-complexity.

In the final part of this work the computational complexity of this model is considered. The main problem from this point of view certainly is to estimate the complexity of finding an optimal circuit representation for a given automaton. Here it is proved that if the automaton is already given in some circuit representation then this problem is PSPACE-complete. PSPACE-complete are also some intuitively simpler problems, for example, to determine for a given automaton whether two of its states are equivalent.

This paper is a summary of the work done by the author that has been published in a series of papers (Valdats, 2011, 2014, 2018).

2 Preliminaries and Notation

It is assumed that the reader is familiar with the basics of the theory of computing: DFA, nondeterministic finite automaton (NFA), regular languages, Turing machine, language complexity classes (PSPACE, NP, ...).

Let $sc(L)$ ($nsc(L)$) denote the state (nondeterministic state) complexity of a regular language L. Let $\mathfrak{L}_s^k$ ($\mathfrak{N}_s^k$) denote the set of all regular languages in a k letter alphabet, whose state (nondeterministic state) complexity does not exceed s.

We will also use Boolean functions and circuits in the standard base $(\&, \vee, \neg)$. The Shannon function is lexicographically the first Boolean function with n inputs and one output whose complexity is maximal (for this n).

For asymptotic comparison we will use the following notation that is taken from (Lupanov, 1984):

$$f(n) \lesssim g(n) \iff \lim_{n\to\infty} \frac{f(n)}{g(n)} \leq 1. \tag{1}$$

This notation is more accurate than the standard big-O notation as it takes into account constant factors. It can be expressed with the standard notation in the following way

$$f(n) \lesssim g(n) \iff f(n) < g(n)(1 + o(1)). \tag{2}$$

For a family of finite sets $\{S_n\}$ we will say that a property $P(x)$ is true for *almost all* $x \in S_n$, if the fraction of x for which $P(x)$ is not true tends to zero when n increases:

$$P(x) \text{ for almost all } x \in S_n \iff \lim_{n\to\infty} \frac{|\{x \in S_n : \neg P(x)\}|}{|S_n|} = 0. \tag{3}$$

3 Encodings and Circuit Representation of DFA

Usually DFA are represented with their state table or their state transition diagram what is essentially the same, the diagram can be thought of as a visualization of the state table. Both these representations depict each state of a DFA separately, therefore with these methods one cannot describe automata with a large number of states.

The s states of the automaton can be encoded into $\lceil \log s \rceil$ state bits. Also the input (and if necessary, also output) symbols can be encoded as bit vectors. Each DFA has many such encodings. If we do not restrict the number of state bits by $\lceil \log s \rceil$ then there are infinitely many.

In such a case the arguments of the transition function of the DFA are an encoded state and an encoded input and it computes an encoded next state value. Represented in this way, it is a Boolean function and a natural way to describe it is with a Boolean circuit.

It is also necessary to describe the set of the accepting states. We will describe it with a circuit that computes the characteristic function of this set.

In such a way we can describe a DFA with encodings of its state set and its input alphabet together with two Boolean circuits F and G: the first for its transition function and the second for the characteristic function of the set of the accepting states. These circuits we will further call *the transition circuit* and *the acceptance circuit* and together they will form the circuit representation of a DFA (F, G).

4 BC-complexity

4.1 BC-complexity of DFA and Regular Languages

Definition 1. The BC-complexity of the circuit representation of a DFA (F, G) is the sum of the complexities of its transition circuit, its acceptance circuit and of the number of state bits:

$$C_{\mathrm{BC}}((F, G)) = C(F) + C(G) + b_Q. \tag{4}$$

Here and further by $C(A)$ we denote the number of gates in the circuit A.

Why do we add the number of state bits b_Q? It ensures that the BC-complexity of arbitrarily large DFAs cannot be zero which will be essential in the further estimations.

The BC-complexity of a DFA is the minimal BC-complexity among all its circuit representations. The BC-complexity of a regular language is the minimal BC-complexity among all DFAs that accept it.

BC-complexity is an abbreviation of Boolean Circuit Complexity and, although it would be nice to call it just circuit complexity, this term in the case of regular languages already has a different meaning.

4.2 Estimation of the lower bound of BC-complexity

Essentially BC-complexity is a circuit complexity and to estimate its lower bound we need to estimate a lower bound of the circuit complexity. It is a well-known hard problem for which the best known lower bounds in the general case are linear.

For this reason usually lower bounds for the circuit complexity are obtained using pigeonhole principle. If the number of circuits with a complexity not exceeding r is less than the size of the given class of Boolean functions, then this class contains a function with complexity larger than r.

For BC-complexity we will use a similar method, all further lower bounds will be obtained using the following theorem:

Theorem 1. *Let $\mathfrak{L}_i$ be a family of sets of increasing size that consist of regular languages in a fixed alphabet Σ. For arbitrary constant a for almost all $L \in \mathfrak{L}_i$*

$$C_{BC}(L) > \frac{\log |\mathfrak{L}_i|}{\log\log |\mathfrak{L}_i| - a}. \tag{5}$$

Proof. Proof is done by the pigeonhole principle. Each regular language has a minimal automaton that accepts it that has a circuit representation. Number of circuit representations with BC-complexity not exceeding $\frac{\log |\mathfrak{L}_i|}{\log\log |\mathfrak{L}_i| - a}$ is much smaller than the size of $\mathfrak{L}_i$ for sufficiently large i. □

4.3 BC-complexity and State Complexity

The following theorem shows that BC-complexity for regular languages with a given state complexity s can differ at most exponentially.

Theorem 2. *For an arbitrary regular language L with state complexity s in a fixed alphabet $|\Sigma| = k$*

$$\lceil \log s \rceil \leq C_{\mathrm{BC}}(L) \lesssim (k-1)s, \qquad \text{if } k \geq 2, \tag{6}$$

$$\lceil \log s \rceil \leq C_{\mathrm{BC}}(L) \lesssim \frac{s}{\log s}, \qquad \text{if } k = 1. \tag{7}$$

Proof. The lower bound. To encode s states we need at least $\lceil \log s \rceil$ state bits, therefore for any circuit representation (F, G) that represents a DFA that accepts L

$$C_{\mathrm{BC}}(F, G) = C(F) + C(G) + b_Q \geq b_Q \geq \lceil \log s \rceil . \tag{8}$$

To prove the upper bound we consider the minimal DFA (with s states) that accepts L. If we choose an arbitrary minimal encoding (with $b_Q = \lceil \log s \rceil$ state bits) and construct a circuit representation using methods for optimal circuit construction we get that its BC-complexity is not larger than ks.

To improve this result from ks to $(k-1)s$ we can choose this minimal encoding in a way that for a specific input symbol the transition function is relatively simple. Short summary of the idea: choose an arbitrary input symbol and consider the state transition graph for this particular symbol. It consists of connected components each of which looks like a "loop" with a possible "tail".

These components can be ordered according to their parameters (length of the loop and the tail) and this ordering of components naturally determines an ordering of states, which we can use as an encoding. For this encoding the complexity of the transition circuit for this input symbol is negligible with a respect to other input symbols, therefore the total BC-complexity decreases from ks to $(k-1)s$. □

Further we consider two particular languages with the same state complexity for whom the difference in BC-complexity is indeed exponential.

The first language L_n in a binary alphabet $\Sigma = \{0, 1\}$ consists of all words for which the n-th last symbol is "1": $w \in L_n \iff w_{|w|-n+1} = 1$. Its state complexity is 2^n, any DFA that recognizes it needs to remember the last n input symbols. But its BC-complexity is n.

The second language L_n^{Sh} consists of words in a binary alphabet, such that the Shannon function returns 1 if applied to their last n bits:

$$w \in L_n^{Sh} \iff Sh_n(w_{k-n+1}, w_{k-n+2}, \ldots, w_k) = 1. \tag{9}$$

One can show that its state complexity does not exceed 2^n but its BC-complexity is at least $2^n/n^2$. The BC-complexity of this language is close to the upper bound of Theorem 2, but still there is a gap. It would be nice to find a language that reaches the bound exactly.

It turns out that almost all languages reach the upper bound of Theorem 2.

Theorem 3. *For almost all languages* $L \in \mathfrak{L}_s^k$

$$C_{\mathrm{BC}}(L) \gtrsim (k-1)s, \qquad \text{if } k \geq 2 \tag{10}$$

$$C_{\mathrm{BC}}(L) > \frac{s}{\log s}, \qquad \text{if } k = 1 \tag{11}$$

Proof. This can be obtained with the aid of some algebra from Theorem 1 and the estimation of $|\mathfrak{L}_s^k|$ that can be found in (Domaratzki et al., 2002). □

4.4 Shannon effect for the BC-complexity

If we look carefully at Theorems 2 and 3 we can notice that the upper bound of the first one is the same as the lower bound for the last. This is called the Shannon effect: For almost all regular languages with a given state complexity their BC-complexity is close to its maximal possible value.

Corollary 1. *For almost all languages* $L \in \mathfrak{L}_s^k$

$$(k-1)s \lesssim C_{\mathrm{BC}}(L) \lesssim (k-1)s, \qquad \text{if } k \geq 2 \tag{12}$$

$$\frac{s}{\log s} < C_{\mathrm{BC}}(L) \lesssim \frac{s}{\log s}, \qquad \text{if } k = 1 \tag{13}$$

5 BC-complexity and nondeterministic state complexity

Knowing that $nsc(L) \leq sc(L) \leq 2^{nsc(L)}$ allows us to use previous estimations of BC-complexity (Theorem 2) to obtain the first "naive" estimations of the BC-complexity of regular languages with respect to their nondeterministic state complexity $s = nsc(L)$:

$$\lceil \log s \rceil \leq C_{\mathrm{BC}}(L) \lesssim (k-1)2^s. \tag{14}$$

The lower bound cannot be improved too much. To show this one can remember that there are languages whose nondeterministic state complexity is the same as their state complexity.

But the upper bound in formula (14) can be improved exponentially as will be shown later. From that we can conclude that in those cases when in the process of determinization the number of states grows exponentially, the resulting automaton has a simple structure and its BC-complexity is relatively small.

We will start with a simple estimation of the BC-complexity of a DFA that is obtained from an NFA using the powerset construction and continue with some improvements that lead to a near-optimal solution.

Theorem 4. *If a regular language* L *in a* k*-letter alphabet can be recognized with an* s*-state NFA with* t *transitions then*

$$C_{\mathrm{BC}}(L) \leq t + (k+1)s. \tag{15}$$

Proof. We construct a circuit representation of a DFA that is obtained from an NFA using the powerset construction. Each state bit in this circuit representation corresponds to one state of the NFA. Transition function in this case is relatively simple and the result can be obtained by carefully counting logic gates. □

As the number of transitions of an NFA does not exceed ks^2 then

Corollary 2. *For all languages in* $L \in \mathfrak{N}_s^k$*:*

$$C_{\mathrm{BC}}(L) \leq ks^2 + (k+1)s. \tag{16}$$

Estimation (15) is good if the number of transitions in the NFA is small (as it is, for example, in the case of a "reversed" DFA), but in the general case it can be improved by an order of $\log s$.

Theorem 5. *For all $L \in \mathfrak{N}_s^k$:*

$$\lceil \log s \rceil \leq C_{\mathrm{BC}}(L) \lesssim \frac{ks^2}{\log s}. \tag{17}$$

Proof. For the upper bound we use the same construction as in Theorem 4, but the construction of the transition circuit is optimized. □

As it was noted in the beginning of the chapter, the lower bound of Theorem 5 is closely reachable. To show that the upper bound also is closely reachable we will act similarly as in the case of state complexity and estimate the BC-complexity of almost all languages with a given nondeterministic state complexity

Theorem 6. *For almost all languages in $L \in \mathfrak{N}_s^k$*

$$C_{\mathrm{BC}}(L) > \frac{(k-1)s^2}{2\log s}, \qquad \textit{if } k \geq 2, \tag{18}$$

$$C_{\mathrm{BC}}(L) > \frac{s}{\log s}, \qquad \textit{if } k = 1. \tag{19}$$

Proof. These estimations with the aid of some algebra can be obtained from Theorem 1 and the estimation of $|\mathfrak{N}_s^k|$ taken from (Domaratzki et al., 2002). □

If we put together the upper and lower bounds (Theorems 5 un 6) we obtain the following.

Corollary 3. *For almost all languages $L \in \mathfrak{N}_s^k$*

$$\frac{(k-1)s^2}{2\log s} < C_{\mathrm{BC}}(L) \lesssim \frac{ks^2}{\log s}, \qquad \textit{if } k \geq 2, \tag{20}$$

$$\frac{s}{\log s} < C_{\mathrm{BC}}(L) \lesssim \frac{s^2}{\log s}, \qquad \textit{if } k = 1. \tag{21}$$

One can see that the upper and lower bounds do not coincide. If $k \geq 2$ they differ by a constant factor (4 if $k = 2$ or less if $k > 2$), but if $k = 1$ the difference is more than by a constant factor. It would be nice to prove the Shannon effect also in this case, but it looks like a hard problem.

6 Language operations

In this chapter we investigate how BC-complexity changes with some language operations: union, intersection, concatenation, Kleene closure and reversal. Table 1 compares the upper bound of the BC-complexity and the state complexity for each of these operations. For all operations $m = sc(L_1)$, $n = sc(L_2)$, $a = C_{\mathrm{BC}}(L_1)$, $b = C_{\mathrm{BC}}(L_2)$, $k = |\Sigma|$.

Table 1. Upper bound of state complexity and BC-complexity for language operations

Operation	State complexity	BC-complexity
$L_1 \cup L_2$	mn	$a+b+1$
$L_1 \cap L_2$	mn	$a+b+1$
L^R	2^m	$(2k+4)m$
$L_1 L_2$	$m2^n - 2^{n-1}$	$a+(2k+3)n$
L_1^*	$2^{m-1}+2^{m-2}$	$(3k+1)m$

One can see that for all the operations the upper bound of the BC-complexity of the obtained language is smaller than the maximal BC-complexity for the corresponding number of states. For example, in the case of concatenation for almost all languages with state complexity $m2^n - 2^{n-1}$ their BC-complexity is around $(k-1)(m2^n - 2^{n-1})$ (Theorem 1) which is much larger than the maximal BC-complexity for languages obtained by concatenation $(a + (2k+3)n)$.

7 BC-complexity and the Minimization of DFA

State assignment problem is deeply connected with state minimization. One can naively think that to obtain the best implementation for a given language one can first minimize the given DFA and then perform the state assignment, but that is not true, not always the minimized DFA will have as good implementation as the original one.

Already in 1962 Hartmanis and Stearns (1962) considered an 8-state DFA which can be realized with 19 diodes, but if it is minimized to 7 states then it needs already 22 diodes (it is not the lower bound but the best implementation known to the authors). Other authors also emphasize that state minimization should not be considered separately from state assignment, but these problems should be addressed synchronously (Calazans, 1993).

It turns out that the BC-complexity of the minimal DFA of a regular language can differ from the BC-complexity of the language itself not only "by 3 diodes" but more than polynomially. Denote by $M(A)$ ($M(L)$) the minimal DFA that is equivalent to A (that accepts L).

Theorem 7. *If there exists a polynomial $p(x)$ such that for all regular languages in a binary alphabet $C_{\mathrm{BC}}(M(L)) < p(C_{\mathrm{BC}}(L))$ then PSPACE$\subseteq$ P/poly.*

Proof. For an arbitrary language $V \in$ PSPACE one can construct a DFA A_n^V that works in a binary alphabet and accepts a word w iff its prefix of length n belongs to V, but it does that after exponentially long time (it ignores the rest of the word). This DFA has a polynomial BC-complexity with respect to n, one can construct it by simulating the Turing machine for V in the state space of the automaton.

If the BC-complexity of the corresponding minimal DFA $M(A_n^V)$ is polynomially bounded by $C_{\mathrm{BC}}(A_n^V)$ then it has a circuit representation whose BC-complexity is polynomial in n.

But the minimal DFA $M(A_n^V)$ will "know" whether the input should be accepted already after the first n input letters, it will be in one state if $w_1 w_2 \dots w_n \in V$ or in

another if $w_1 w_2 \dots w_n \notin V$. Using this fact one can construct a circuit that recognizes the words of length n of the language V and this circuit will have a polynomial (in n) complexity. □

Karp-Lipton theorem says that if NP⊆P/Poly then the polynomial hierarchy collapses to its second level Σ_P^2. Also it is well known that if PSPACE $\subseteq$ P/poly then PSPACE$\subseteq \Sigma_2^P \cap \Pi_2^P$. Therefore it is widely assumed (although not proved) that PSPACE$\not\subseteq$ P/poly is true.

8 Computational Complexity for DFA in Circuit Representation

If we consider problems like state equivalence, DFA equivalence or minimization then for DFA in standard representation they all are solvable in polynomial time. Because of this simplicity their complexity analysis is rarely mentioned in the literature with the exception of DFA minimization which is a bit harder problem than the others and also significant in practice.

But the situation changes completely if we consider these problems for DFA given in their circuit representation. It turns out that in such a case all these problems are PSPACE-complete.

At first we will show that state reachability for DFA in circuit representation is PSPACE-complete. Denote by REACH$_{CR}$ the language consisting of all pairs $(s, (F, G))$ where (F, G) is a circuit representation of a DFA and s is a state that is reachable in it.

Theorem 8. *For a fixed input alphabet, REACH$_{CR}$ is PSPACE-complete.*

Proof. To prove this we will use the theory of succinct representation of algorithmic instances that was developed in the 80-s and 90-s (Borchert and Lozano, 1986). Two basic concepts of this theory are succinct instances and polylogarithmic time reduction.

The hardest part of the proof is to show that REACH$_{CR}$ is PSPACE-hard. At first one can show that state reachability for DFA in standard representation is NL-complete under polylogarithmic time reduction. Then using the theory of succinct representations of algorithmic instances one can conclude that state reachability for DFA in a succinct representation is PSPACE-hard. And finally the circuit representation of a DFA can be obtained in polynomial time from its succinct representation. □

Using this result we can show that many more problems for DFA in circuit representation are PSPACE-complete.

Theorem 9. *The following problems are PSPACE-complete:*

1. *Given a circuit representation of a DFA and (an encoding of) two of its states, determine whether these states are equivalent.*
2. *Given a circuit representation of a DFA, determine whether the language it accepts is the empty language.*
3. *Given two circuit representations of DFAs, determine whether these DFAs are equivalent.*
4. *Given a circuit representation of a DFA and a number k, determine whether there is an equivalent DFA with BC-complexity at most k.*

Proof. For each of these problems one can relatively easily show that they are in PSPACE and for the hardness we can use reduction. The first problem can be reduced to Theorem 8, the third and the second ones can be reduced to the first one and the fourth one can be reduced to the third one. □

The last problem in Theorem 9 is the optimal circuit representation problem for DFA stated as a decision problem. It shows that despite the fact that the minimal DFA can easily be found in polynomial time, finding the optimal circuit representation is PSPACE-complete.

But this is not the most natural setting of the problem. If we return to the state assignment problem then usually the question asked is: find an optimal (minimal) circuit representation for a DFA given in the standard (state table) representation. If we look at this problem formally then it actually can be set in two slightly different variants:

1. Given DFA in a standard representation determine if its BC-complexity is less than a given integer k.
2. Given DFA in a standard representation determine if the BC-complexity of the language it accepts is less than a given integer k.

In the first case we need to find a minimal circuit representation for a given DFA, but in the second case we can take any equivalent DFA as well. Both these problems are in PSPACE but at least for the first problem one can relatively easily prove that it is in NP.

At the first moment it seems that we can prove the same also for the second formulation. Indeed, we can nondeterministically guess an equivalent DFA, its encoding and circuit representation and then in polynomial time check their equivalence and also check that this guessed circuit representation represents this DFA. But unfortunately we cannot guarantee that the number of states of this equivalent DFA is polynomially bounded.

Thereby the question how hard is the state assignment problem turns out to be quite nontrivial. In the first formulation (when we want to find a circuit representation for a particular DFA) it belongs to NP, but there is no evidence about its NP-completeness, rather opposite, there is some evidence that it is not the case (Kabanets and Cai, 2000). In the second formulation this question is even harder, we even do not know if it is in NP.

In the end we can note that in the case when we are given an NFA instead of a DFA then the optimal circuit representation problem is PSPACE-complete. Indeed, it is well known that to determine whether a given NFA is equivalent to the all-accepting DFA is a PSPACE-complete problem (Garey and Johnson, 1979). That means that our problem in the case of NFA is PSPACE-complete even for $k = 0$.

9 Conclusions

The main subject of this work is the BC-complexity of DFA and regular languages what can be interpreted as a formalism for the state assignment problem of DFA. Although

its definition is simple and one could have analyzed it already in the 60-s, this question until now has not been considered from the complexity point of view.

BC-complexity characterizes the internal structure of the DFA. In case if a DFA has no internal structure its BC-complexity is close to its state complexity but in case it has it can be even exponentially smaller (Theorem 2). So called Shannon effect for the BC-complexity tells that almost all regular languages with a given state complexity have "no internal structure", their BC-complexity is close to the maximum.

The same question for the nondeterministic state complexity is considered in Chapter 5. Upper and lower bounds for the BC-complexity with a respect to the nondeterministic state complexity are quite close (they differ 4 times in the case of a binary alphabet), but do not coincide. Therefore the question about the Shannon effect for languages with a given nondeterministic state complexity remains open.

The upper bounds for the BC-complexity of some language operations have been estimated. All of them are smaller than the BC-complexity for almost all languages with the corresponding state complexity that tells us that they are described by automata "with internal structure".

One of the most interesting results of this work can be found in Chapter 7. It proves in a strong form the well known fact that the minimization of a DFA can lead to the loss of its internal structure. Until now it was justified only by some small examples, but Theorem 7 says that the BC-complexity of the minimal DFA of a regular language is not even polynomially bounded with the BC-complexity of the language itself (supposing PSPACE$\not\in$ P/Poly).

But another problem which in some sense is similar to the previous one remains unsolved: can we decrease the BC-complexity for a DFA if we use an encoding which is not minimal. An n-state DFA can be encoded into $\lceil \log n \rceil$ state bits. Could it happen that the minimal circuit representation of this DFA has more than $\lceil \log n \rceil$ state bits?

Many problems that are so easy for DFA given in the standard representation that their complexity analysis is hard to find in the literature (e.g. state reachability, equivalence) turn out to be much harder for DFA given in their circuit representation. A few of such problems are shown to be PSPACE-complete in Chapter 8 including the problem of finding the optimal circuit representation of a DFA given in circuit representation. But for a DFA given in the standard representation (what is probably the most practical problem statement) the complexity of this problem remains unknown.

References

Borchert, B., Lozano, A. (2007). Succinct circuit representations and leaf languages are basically the same concept. *Universitat Politechnica de Catalunya*, technical report, available at `http://upcommons.upc.edu/handle/2117/97245`

Calazans, N. L. V. (1993). Considering state minimization during state assignment, In: *I Ibero American Microelectronics Conference-X Congress of the Brazilian Microelectronics Society*, 49–58.

De Micheli, G., Brayton, R., Sangiovanni-Vincentelli, A. (1985). Optimal state assignment for finite state machines, In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 4(3), 269–285.

Domaratzki, M., Kisman, D., Shallit, J. (2002). On the number of distinct languages accepted by finite automata with n states, In: *Journal of Automata, Languages and Combinatorics*, 7(4) 469–486.

Garey, M. R., Johnson, D. S. (1979). *Computers and intractability*, Freeman, New York.

Hartmanis, J., Stearns, R. E. (1962). Some dangers in state reduction of sequential machines, In: *Information and Control*, 5(3), 252–260.

Kabanets, V., Cai, J. (2000). Circuit minimization problem, In: *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, 73–79.

Kajstura, K., Kania, D. (2017). Low Power Synthesis of Finite State Machines — State Assignment Decomposition Algorithm, In: *Journal of Circuits, Systems and Computers*, 1850041.

Kohavi, Z., Jha, N. K. (2009). *Switching and finite automata theory*, 3rd ed., Cambridge University Press.

Lupanov, O. (1984). *Asymptotic Estimates of the Complexity of Control Systems* (Russian), Moscow State University.

Valdats, M. (2011). Transition function complexity of finite automata, In: *International Workshop on Descriptional Complexity of Formal Systems*, 301–313.

Valdats, M. (2014). Boolean Circuit Complexity of Regular Languagues, In: *International Conference on Automata and Formal Languages*, 151, 342–354.

Valdats, M. (2018). Descriptional and Computational Complexity of the Circuit Representation of Finite Automata, In: *International Conference on Language and Automata Theory and Applications*, 105–117.

Received July 24, 2019 , accepted August 3, 2019