Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 321 (2016) 23–39

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

On the Real-state Processing of Regular Operations and The Sakoda-Sipser Problem

J. Andres Montoya[1,2] David Casas[3]

Departamento de Matemáticas
Universidad Nacional de Colombia
Bogotá, Colombia

Abstract

In this work we study some aspects of state-complexity related to the very famous Sakoda-Sipser problem. We study the state-complexity of the regular operations, we survey the known facts and, by the way, we find some new and simpler proofs of some well known results. The analysis of the state of art allowed us to find a new and meaningful notion: Real-state processing. We investigate this notion, looking for a model of deterministic finite automata holding such an interesting property. We establish some preliminary results, which seem to indicate that there does not exists a model of deterministic finite automata having real-state processing of regular expressions, but, on the other hand, we are able of exhibiting a deterministic model of finite automata having real-state processing of star free regular expressions.

Keywords: Finite automata, State complexity, Regular expressions, Regular operation.

1 Introduction

It is known that nondeterministic finite state automata (1NFAs) are as powerful as deterministic finite state automata (1DFAs), in the sense that 1NFAs can only recognize regular languages. It is also known that 1NFAs are more powerful than 1DFAs, because 1NFAs cannot be simulated by 1DFAs with a polynomial overhead in the number of states. Sakoda and Sipser [8] asked if 1NFAs can be simulated by two-way deterministic finite state automata (2DFAs) with a polynomial overhead in the number of states. It is one of the questions included in the, so called, *Sakoda-Sipser Problem.* The Sakoda-Sipser question is a question about: how, when and to which extent can two-wayness replace nondeterminism? It would be great news if such a question would have an affirmative answer. It is the case, given

[1] The first author would like to thank Universidad Nacional de Colombia and the support provided through the R.P. Hermes 16860.

[2] Email: jamontoyaa@unal.edu.co

[3] Email: dfcasast@unal.edu.co

http://dx.doi.org/10.1016/j.entcs.2016.02.003

that 1NFAs (and 2NFAs) are unreliable automata which cannot be used in practice. But, in despite of their *purely theoretical value*, 1NFAs have some remarkable features, which we would like to have in some model of reliable and implementable finite state automata. Thus, we think that the Sakoda-Sipser question is a special case of the following more general question: how, when and to which extent can deterministic finite state automata with added abilities be as powerful and efficient as their nondeterministic counterparts?

Before attacking the later question we will have to consider the following one: which are those remarkable features of nondeterministic finite automata? We won't provide an exhaustive list of remarkable characteristics, but we would like to point out, and to discuss, below, one of those features which has captured our attention.

If one is asked to prove that the set of regular languages is closed under the regular operations, it is very easy to figure out such a proof, if one is allowed to use 1NFAs. Things become harder if one is obligated to employ, in the proof, the weaker model of 1DFAs. Moreover, such an easy proof using 1NFAs yields a linear time algorithm, called Thompson's algorithm, which, on input α (where α is a regular expression), computes an $O(|\alpha|)$-state 1NFA recognizing the language $L(\alpha)$.

We say that a model of deterministic finite automata has *Thompson property*, if and only if, there exists a polynomial time algorithm, which, on input α, computes an $O(|\alpha|^c)$-state automaton within the model, and which recognizes the language $L(\alpha)$, (where c is some fixed constant).

Thus, we have that Thompson property holds for 1NFA, while it is very easy to prove it does not hold for 1DFA. We consider that Thompson property is a remarkable feature of 1NFAs, given that it allows those automata to efficiently process regular expressions. Take into account that the processing of regular expressions is (one of) the main task(s) assigned to finite automata. Unfortunately, the nondeterministic nature of 1NFAs makes them become a nonimplementable solution to the aforementioned problem. Thus, it would be great news if we could exhibit a deterministic model of automata for which Thompson property holds.

In this work we investigate the following question: how, when, and to which extent is it possible to define a model of deterministic finite automata for which Thompson property holds?

Remark 1.1 We understand the Sakoda-Sipser problem as the question: does there exist a deterministic model of finite state automata which can efficiently simulate nondeterministic automata? It is clear that a positive answer to Sakoda-Sipser implies that our problem can be positively solved. On the other hand, if we could give a positive answer to our question, we could not immediately conclude that Sakoda-Sipser also has a positive answer. It is the case because nondeterministic automata are exponentially more succinct than regular expressions [4].

Remark 1.2 We assume that the reader knows the definition of the basic models of finite state automata such as DFAs, NFAs, 2DFAs an so on. The interested reader can consult the excellent reference [9].

Organization of the work and contributions. This work is organized into six sections. In Section 1 we introduce the problem that we study in this paper, introducing the notion of real-state processing. In Section 2 we consider the model of DFAs, and we present simpler proofs of some well known results. In Section 3 we study 2DFAs, and we prove that those automata do not have real-state processing of regular operations. Moreover, we prove a strict superpolynomial separation with respect to the model of 1NFAs. In Section 4 we consider a model of two-way pebble automata, and we prove that it is not able of real-state processing concatenations. In Section 5 we introduce a new model of multiplebble automata, and we prove that it has real-state processing of star-free regular expressions. We conclude, in Section 6, with some concluding remarks.

2 Nondeterministic Finite State Automata and Thompson Property

A model of finite automata is an infinite set $\mathcal{C}$ of state-based recognition devices (automata), that accepts the regular languages, it means that given a regular language there must exist an automaton in $\mathcal{C}$ recognizing the language, and given $\mathcal{M} \in \mathcal{C}$, the language recognized by $\mathcal{M}$ is regular.

We are interested in standard models of finite automata, whose members are constituted by a finite set of internal states, a transition function and, perhaps, some other (finite) resources. For all those models the following definition makes sense.

Definition 2.1 We say that a model of finite automata, say $\mathcal{C}$, solves the problem of efficiently processing the regular expressions if and only if there exists a polynomial time algorithm $\mathcal{T}$, which, on input α (where α is a regular expression), computes $\mathcal{M}_\alpha \in \mathcal{C}$ such that:

- $L(\mathcal{M}_\alpha) = L(\alpha)$.
- $|Q(\mathcal{M}_\alpha)| \in O(|\alpha|^c)$, where $Q(\mathcal{M}_\alpha)$ denotes the set of internal states of automaton $\mathcal{M}_\alpha$, the symbol $|\alpha|$ denotes the length of α, and c is some positive constant.

Does there exist a model of finite automata that solves the problem of efficiently processing the regular expressions? Yes, there is at least one such model, it is the model of 1NFAs. There exists a linear time algorithm $\mathcal{T}$ which, on input α, outputs $\mathcal{M}_\alpha$, a 1NFA such that $L(\alpha) = L(\mathcal{M}_\alpha)$, and such that $|Q(\mathcal{M}_\alpha)| \in O(|\alpha|)$. Algorithm $\mathcal{T}$ is known as *Thompson's Algorithm* [10], and it is a naive algorithm that exploits the recursive definition of regular expressions plus the following crucial fact: there exists three constants C_U, $C.$ and C_*, such that given two 1NFAs, say $\mathcal{M}$ and $\mathcal{N}$, one can compute in linear time three 1NFAs $\mathcal{S}^\cup, \mathcal{S}^\cdot$ and $\mathcal{S}^*$ such that:

(i) $\mathcal{S}^\cup$ accepts the language $L(\mathcal{M}) \cup L(\mathcal{N})$, and the number of its states is equal to $n + m + C_\cup$.

(ii) $\mathcal{S}^\cdot$ accepts the language $L(\mathcal{M}) \cdot L(\mathcal{N})$, and the number of its states is equal to $n + m + C..$

(iii) $\mathcal{S}^*$ accepts the language $L(\mathcal{M})^*$, and the number of its states is equal to $n + C_*$.

It is easy to check that the above three facts guarantee that the output of Thompson algorithm, which is the automaton $\mathcal{M}_\alpha$, satisfies the condition $|Q(\mathcal{M}_\alpha)| \in O(|\alpha|)$. It is also easy to check that weaker conditions (as for example $|Q(\mathcal{S}^\cup)| \leq 2n + m + C_\cup$) are not enough to guarantee the existence of a *Thompson Algorithm.* We say that *Thompson Property* holds for $\mathcal{C}$, if and only if, a naive algorithm exploiting the recursive definition of regular expressions can be put to work within the *state constraint* $|Q(\mathcal{M}_\alpha)| \in O(|\alpha|^c)$. For which models of automata does Thompson property holds? We wont characterize those models, but, to begin with, we will exhibit a condition that guarantees that this elusive property actually holds.

Definition 2.2 We say that $\mathcal{C}$ has real-state processing[4] (real-state conversion) of regular operations if and only if there exists three constants C_U, $C_\cdot$ and C_* such that given two $\mathcal{C}$-automata, say $\mathcal{M}$ and $\mathcal{N}$, with m and n states (respectively), one can compute in polynomial time three $\mathcal{C}$-automata $\mathcal{S}^\cup, \mathcal{S}^\cdot$ and $\mathcal{S}^*$ such that:

(i) $\mathcal{S}^\cup$ accepts the language $L(\mathcal{M}) \cup L(\mathcal{N})$, and the number of its states is equal to $m + n + C_\cup$.

(ii) $\mathcal{S}^\cdot$ accepts the language $L(\mathcal{M}) \cdot L(\mathcal{N})$, and the number of its states is equal to $m + n + C_\cdot$.

(iii) $\mathcal{S}^*$ accepts the language $L(\mathcal{M})^*$, and the number of its states is equal to $m + C_*$.

We have

Proposition 2.3 *If $\mathcal{C}$ has real-state processing of the regular operations, then Thompson property holds for $\mathcal{C}$*

From now on, we will be studying the following problem

Problem 2.4 ***(The Real-State Processing Problem)*** *Does there exists a deterministic model of finite automata which has real-state processing of the regular operations?*

3 Deterministic One-way Finite Automata

The first model of deterministic automata that we will consider is the standard model of 1DFAs. We count, in this model, with a powerful tool for lowerbounding the state complexity of a given regular language, it is Myhill-Nerode theorem.

Definition 3.1 Given an alphabet Σ, and given a language $L \subset \Sigma^*$, we define an equivalence relation $R_L \subset \Sigma^* \times \Sigma^*$ in the following way: given $x, y \in \Sigma^*$, we have that xR_Ly if and only if for all $w \in \Sigma^*$ it happens that $xw \in L \iff yw \in L$.

Theorem 3.2 ***(Myhill-Nerode)***

[4] In analogy with the notion of real-time

$L \subset \Sigma^*$ is a regular language if and only if the quotient $\frac{\Sigma^*}{R_L}$ is finite. Moreover, if the language L is regular, then a minimal 1DFA recognizing L has $|\frac{\Sigma^*}{R_L}|$ states.

For a proof see [9].

We can use Myhill-Nerode theorem to lowerbound the state-complexity of unions in this model. Next result is part of the folklore of automata theory, but the proof of the lower bound is neither trivial nor easy to find in the standard references, we include it for the sake of completeness.

Lemma 3.3 *(upper and lower bounds for unions)*

(i) *Let $\mathcal{N}$ be a 1DFA with n states, and let $\mathcal{M}$ be a 1DFA with m states (over the same input alphabet), the language $L(\mathcal{N}) \cup L(\mathcal{M})$ can be recognized by an 1DFA with at most nm states.*

(ii) *There exists a sequence of regular languages, say $(P_n)_{n\geq 1}$, such that for all $n \geq 1$, the language P_n can be accepted by an 1DFA with n states, but such that for infinitely many pairs (n, m), the language $P_n \cup P_m$ requires nm states.*

Proof. Given $\mathcal{N}$ and $\mathcal{M}$, like in the statement of item 1, we define a third automaton $\mathcal{U}$ as follows:

- $Q_{\mathcal{U}} = Q_{\mathcal{N}} \times Q_{\mathcal{M}}$.
- $q_{0_{\mathcal{U}}} = (q_{0_{\mathcal{N}}}, q_{0_{\mathcal{M}}})$.
- $\delta_{\mathcal{U}} : (Q_{\mathcal{N}} \times Q_{\mathcal{M}}) \times \Sigma \to Q_{\mathcal{N}} \times Q_{\mathcal{M}}$ is defined in the following way.

$$\delta_{\mathcal{U}}((q_{\mathcal{N}}, q_{\mathcal{M}}), x) = (\delta_{\mathcal{N}}(q_{\mathcal{N}}, x), \delta_{\mathcal{M}}(q_{\mathcal{M}}, x)).$$

- $A_{\mathcal{U}} = (A_{\mathcal{N}} \times Q_{\mathcal{M}}) \cup (Q_{\mathcal{N}} \times A_{\mathcal{M}})$.

Automaton $\mathcal{U}$ has nm states, and it is easy to check that it recognizes the language $L(\mathcal{N}) \cup L(\mathcal{M})$.

Given $n \geq 1$, we set

$$P_n = \{x \in \Sigma^* : |x| \equiv 0\,(mod\,(n))\}$$

We have that $x \equiv_{P_n} y$ if and only if $|x| \equiv |y|$ $(mod\,(n))$, and hence we know that there exists a 1DFA with n states recognizing the language P_n. Now, we consider the sequence $\{P_n^c\}_{n\geq 1}$. Given $n \geq 1$, there exists a 1DFA with n states recognizing the language P_n^c. We can use Chinese remaindering to prove that $P_n^c \cup P_m^c$ requires $\text{lcm}(n, m)$ states. Now, if we suppose that n and m are *coprime* (i.e. $\gcd(n, m) = 1$) we get the lower bound nm. □

Corollary 3.4 *1DFAs do not have real-state processing of unions.*

We know that real-state processing implies Thompson property, but given that the converse is not true it is still possible that Thompson property holds for 1DFAs. It is easy to prove that it is not the case.

Proposition 3.5 *Thompson property does not hold for 1DFAs.*

Proof. Suppose that Thompson property holds for 1DFAs, then given a regular expression α there must exist a 1DFA with $O(|\alpha|)$ states and which recognizes the language $L(\alpha)$. We know that it is not possible because regular expressions are *exponentially more succinct* than 1DFAs. □

We used in the above proof that regular expressions are exponentially more succinct than 1DFAs, it means that there exists a sequence of regular languages $\{L_k\}_{k\geq 1}$, and there exists a constant $C > 1$ such that:

- For all k, there exists a regular expression α_k which denotes the language L_k and whose length is linear in k.
- Given $k \geq 1$, a minimal 1DFA accepting L_k requires $\Omega\left(C^k\right)$ states.

There are many examples of sequences that behave this way, we include a classical example, which will be used again in the next sections. Let $\Sigma = \{0, 1\}$, let $k \geq 1$ and let L_k be the language defined by

$$L_k = \{x \in \Sigma^* : x[|x| - k + 1] = 1\}$$

That is: language L_k is the set of all strings such that the position that is placed at k positions from the right end is filled with a 1.

Let $k{\geq}1$, it is easy to check that a regular expression denoting the language L_k is the expression $(0 \cup 1)^* 1 (0 \cup 1)^{k-1}$, notice that the length of this expression is equal to $5 + (k - 1)3$. Now consider the equivalence relation $\equiv_k$ determined by the language L_k. We have that x, y are in the same equivalence class if and only if the last k characters of both strings are all equal. Then, we can claim that there are at least 2^k equivalence classes, and it implies that a minimal 1DFA recognizing L_k has at least 2^k states.

Thus, we know that the model of 1DFAs is not the right model, Thompson property does not hold for it, given that it is not able of real-state processing unions (which seems to be the more tractable of the regular operations) and, as we will see, it behaves even worse when it comes to the processing of concatenations.

Next result is a well known result [11], but we have found a new proof which seems to be very much simpler.

Theorem 3.6 *The 1DFA-state complexity of concatenation is at least exponential.*

Proof. Let $\Sigma = \{0, 1\}$, and let $\{A_n\}_{n\geq 1}$ be the sequence of languages defined by:

$$A_n = \{x \in \Sigma^* : |x| \geq 0\}$$

Notice that $\{A_n\}_{n\geq 1}$ is a constant sequence (all the languages are the same). Finally, we introduce a second sequence $\{B_n\}_{n\geq 1}$, where given $k \geq 1$ the language B_k is equal to

$$\{x \in \Sigma^* : x[1] = 1 \;\&\; |x| = k\}$$

We can check that for each n, and for each k the equality $A_n \cdot B_k = L_k$ holds. It is also easy to check that:

(i) For all n, language A_n can be recognized using an automaton with an unique internal state.

(ii) For all n, language B_n can be recognized using an automaton with $n+2$ states.

(iii) For all n, language L_n requires 2^n states.

Altogether, we get an exponential lower bound for the processing of concatenations employing 1DFAs. □

4 Deterministic Two-Way Finite State Automata

In this section we study the model of *two-way terminating deterministic finite state automata* (2DFAs, for short).

Let $\mathcal{M} = (Q, \Sigma, q_0, F, \delta)$ be a 1DFA, and let $x = x_1x_2 \dots x_n$ be a string of size n. The computation of $\mathcal{M}$, on input x, takes n time units which is the time required by the workhead to reach the right end of the input. Now suppose that $\mathcal{M}$ is a two-way deterministic finite automaton, the computation of $\mathcal{M}$, on input x, could be infinite. We will avoid this possibility restricting ourselves to studying two-way terminating finite state automata, which are the two-way deterministic finite automata that halt on all their inputs. There is not loss of generality if we restrict the investigation to the later type of two-way automata, it is the case given that:

(i) There is no real loss of computation power: the restricted model of two-way terminating automata can recognize all the regular languages (any 1DFA is a 2DFA which never moves leftward).

(ii) There is not a significant blow-up in state-complexity: any two-way deterministic automaton with n states can be simulated by a 2DFA with $4n+1$ states [5].

It is known that 2DFAs are exponentially more powerful than 1DFAs. To check this, it is enough to consider the sequence $\{L_n\}_{n\geq 1}$ introduced in the proof of theorem 3.6, notice that L_n can be recognized employing a 2DFA that uses $n+3$ states, while any 1DFA recognizing L_n requires 2^n states. Interesting enough, it can be proved that 2DFAs could be exponentially more powerful than 1NFAs. This fact was known by Sakoda and Sipser [8], who proved the result using a sequence of languages which has been instrumental in the study of The Sakoda-Sipser problem (the sequence defining *The Liveness Problem* [8]). We will include a proof of this fact, which is based on a very much simpler sequence of languages.

Theorem 4.1 *There exists a sequence of regular languages, say $\{M_n\}_{n\geq 1}$, such that.*

(i) *Given $n \geq 1$, there exists a 2DFA recognizing M_n which uses at most $4n+3$ states.*

(ii) *Given $n \geq 1$, it happens that any 1NFA recognizing M_n requires at least 2^n states.*

Proof. Given $n \geq 1$, we set $\Sigma_n = \{1, \dots, n\}$. Given $P = \{i_1, ..., i_k\} \subseteq [n]$, we define

$L_P^n = \{i_1, ..., i_k\}^*$. Finally, given $n \geq 1$, we define M_n as the language

$$\left\{w \in \Sigma_n^* : \exists P\left(w \in L_P^n 0 L_{[n]-P}^n\right)\right\}$$

It is not hard to figure out a terminating 2DFA with $4n+3$ states recognizing the language M_n. Now, we will prove that any 1NFA recognizing M_n requires 2^n states. Each $P \subseteq \Sigma_n$ can be represented by a string w_P which corresponds to write down the elements of P in increasing order. Thus, we have 2^n different pairs of strings, the pairs in the set $\left\{(w_P 0, w_{[n]-P}) : P \subseteq [n]\right\}$, satisfying the following two conditions:

(i) $w_P 0 w_{[n]-P} \in M_n$.

(ii) If $P \neq Q$, the string $w_P 0 w_{[n]-Q}$ does not belong to M_n.

The existence of such a set of pairs, of size 2^n, implies that any 1NFA recognizing the language M_n has at least 2^n states [1]. □

Problem 4.2 *Our proof, as well as Sakoda-Sipser's proof, employs increasing alphabets. It is natural to ask if the same exponential separation can be achieved over a fixed alphabet. Does a similar separation hold in the unary case?*

Thus, the model of 2DFAs seems to be powerful enough as to be able of efficiently simulating nondeterministic finite automata. Do 2DFAs have real-state processing of regular operations?

Theorem 4.3 *Let $\mathcal{M}_1$ be a 2DFA with m states and let $\mathcal{M}_2$ be a 2DFA with n states, there exists a 2DFA that recognizes the language $L(\mathcal{M}_1) \cup L(\mathcal{M}_2)$, using no more than $m+n+1$ states.*

Proof. Suppose that Q_i is the set of states of $\mathcal{M}_i$, $i = 1, 2$. We claim that one can construct a 2DFA $\mathcal{N}$ which recognizes the language $L(\mathcal{M}_1) \cup L(\mathcal{M}_2)$, and such that the set of its states is equal to $Q_1 \cup Q_2 \cup \{q\}$ (we can suppose, without loss of generality, that Q_1 and Q_2 are disjoint sets and that $q \notin Q_1 \cup Q_2$). The computation of this automaton begins in the initial state of $\mathcal{M}_1$, and it proceeds by simulating $\mathcal{M}_1$ until a final state is reached (either an accepting or a rejecting state), if the final state reached is an accepting state, automaton $\mathcal{N}$ halts and accepts the input, otherwise it enters the special state q, and begins to look for the left end of the input. Once the left end is reached, automaton $\mathcal{N}$ begins to simulate the computation of $\mathcal{M}_2$. Along this second phase a final state must be eventually reached, if this final state is accepting the automaton accepts, otherwise it rejects the input. □

Now, we consider the concatenation operation. Next result is taken from [11]:

Theorem 4.4 *Let $m, n \geq 1$, the language $T_m = L\left(a^{m-1}(a^m)^*\right)$ can be recognized by an m-state 1DFA, but if n and m are coprime the concatenation language $T_n \cdot T_m$ requires mn states.*

The above lower bound indicates that 2DFA are unable of real-state processing concatenations, even in the unary case. Thus, it seems that real-state processing is something that is very hard to achieve. We will relax our problem a little bit, considering the following weaker question:

Problem 4.5 ***(The real-state processing problem for star free expressions)*** *Does there exist a model of deterministic finite automata which has real-state processing of star free expressions?*

We will investigate this new problem in the remaining of the paper, but before of this we will discuss the existence of a superpolynomial separation (related to problem 4.2) that holds in the unary case.

Definition 4.6 Landau's function
Landau's function is the function $g : \mathbb{N} \to \mathbb{N}$ defined by

$$g(n) = max\{lcm(p_1, \ldots, p_k) : k \geq 1 \ \& \ p_1 + \cdots + p_k \leq n\}$$

Landau introduced this function in the study of some number theoretical problems, he proved the asymptotic lower bound $g(n) \geq e^{(1+o(1))\sqrt{nln(n)}}$. Next result is taken from [11].

Theorem 4.7 *The language $R_n = L\left(a^{g(n)-1}(a^{g(n)})^*\right)$ can be recognized employing a n-state 2DFA, but every 2DFA accepting R_n^* has at least $(g(n)-1)^2$ states.*

The first corollary that we could get from the above theorem is that 2DFAs cannot real-state process the Kleene star. We can get a second interesting corollary, which gives a definitive answer to problem 4.2.

Corollary 4.8 *Unary 2DFAs are superpolynomially more succinct than unary 1NFAs.*

Proof. Let $h_1(n)$ be the number of states of a minimal 1NFA recognizing the language R_n, and let $h_2(n)$ be the number of states of a minimal 1NFA recognizing the language R_n^*. Notice that $h_1(n) = h_2(n)$. We define $g_1(n)$ as the number of states of a minimal 2DFA recognizing the language R_n, and we define $g_2(n)$ as the number of states of a minimal 2DFA recognizing the language R_n^*. Notice that

$$g_1(n) \leq n < e^{(1+o(1))\sqrt{nln(n)}} \leq g_2(n)$$

Now, we use that any n-state unary 1NFA can be simulated by a 2DFA with a (at most) quadratic overhead in the number of states [3]. Thus, we have that

$$h_1(n) \geq e^{\frac{(1+o(1))\sqrt{g_1(n)\ln(g_1(n))}}{2}}$$

It is clear that a function $e^{\frac{(1+o(1))\sqrt{m\ln(m)}}{2}}$ is superpolynomial in m, and then the corollary is proved. □

Remark 4.9 Corollary 4.8 shows that, in the unary world, 2DFAs behaves better than 1NFAs when it comes to the processing of regular operations. It should not be considered a big surprise: recall that 2DFAs behaves better than 1NFAs when it comes to the processing of the intersection and complementation operations (see [9]), which are nonregular operations that are analogous to the three basic regular operations.

5 Deterministic Two-Way Pebble Automata

Star free regular expressions are the regular expressions that can be constructed using only unions and concatenations, these expressions constitute an important class of expressions: these are the regular expressions that denote the finite languages. One could think that finite languages are boring, but he has take into account that these are the formal languages that are most used in the theory of programming languages (the area where the efficient processing of regular expressions becomes an important task). Does there exist a deterministic model of finite automata which has real-state processing of the star free regular expressions?

The models of automata studied so far have several different features but a common feature: they cannot write on their tapes. If we add those automata the ability of writing on their tapes, we could leave the regular world. There exists some very weak forms of writing, which does not force 2DFAs to leave the regular world, one important example is the writing ability provided by a single pebble. Ibarra et al [2] studied a model of pebble automata that accepts the regular languages, Ibarra automata are two-way deterministic automata provided with a single pebble which is used by those automata to mark cells on their tapes (for definitions see [2]). We use the symbol 1p2DFA to denote the class of Ibarra automata that adhere to the following further restriction:

Let $\mathcal{M}$ be a 1p2DFA. It has two initial states, the *authentic initial* state which is called the L-initial state, and a second special state which is called the R-initial state. Suppose that cell i is the current position of the pebble, and suppose that the workhead is located on cell j, with $j < i$. Then, the workhead is forced to stay within the first $i - 1$ cells until the automaton reaches a *transition state*. Transition states are divided into accepting and rejecting states. Once a transition-accepting state is reached, the automaton looks for the pebble, locates it without picking it up, changes its state to the R-initial state and begins to work on the right side of the tape. On the other hand, if a transition-rejecting state is reached, the automaton looks for the pebble, picks it up, advances one step to the right, places the pebble on the next to the right cell, looks for the leftend and changes its initial state to the L-initial state. The pebble can also be picked up from the right, but it can happen only if the automaton has reached a transition-rejecting state while being on this side of the tape. Once the automaton reaches a transition-accepting state, while being working on the right, it halts and accepts the input. On the other hand, if the transition state is rejecting, the automaton looks for the pebble, locates it and picks it up, moves one step to the right, places the pebble on this cell (the next to the right cell), looks for the left end of the tape and changes its internal state to the L-initial state. Thus, our 1pDFAs use the pebble only to cut the tape into two disjoint segments. When the pebble is placed on the tape, those automata work first on the left segment, and then on the right segment. Moreover, their computations are divided into completely independent stages, the transition between two successive stages being given by moving the pebble one step to the right. Notice that those automata are tailor-made to process the concatenation of two regular languages.

Our restricted model of pebble automata accepts the regular languages: this model cannot accept nonregular languages because it is weaker than the Model of Ibarra et al, and, on the other hand, the model can accept all the regular languages because a 2DFA is a 1p2DFA that never uses its pebble.

It is very easy to check that 1p2DFAs are able of real-state processing unions. It could happens that the pebble (this new ability) allows those automata to real-state process concatenations, but first we have to ask: do those very restricted pebbles yield some computation power?

Definition 5.1 Given $\mathcal{C}$ and $\mathcal{D}$, two different models of automata, we say that $\mathcal{C}$ cannot be linearly-simulated by $\mathcal{D}$ if and only if there exist a sequence of regular languages, say $\{U_n\}_{n\geq 1}$, a function $f : \mathbb{N} \to \mathbb{N}$ and a positive real number ε such that for all n there exists a $\mathcal{C}$-automaton with $f(n)$ states recognizing the language U_n, while any $\mathcal{D}$-automaton recognizing the same language requires $\Omega\left(f(n)^{1+\varepsilon}\right)$ states.

We prove that 1p2DFAs cannot be linearly-simulated by 2DFAs, even in the unary case

Theorem 5.2 *1p2DFAs cannot be linearly-simulated by 2DFAs, even in the unary case.*

Proof. Suppose that we have two regular languages, say L and T, and suppose that we have a n-state 2DFA $\mathcal{N}$ recognizing the language L, and a m-state 2DFA $\mathcal{M}$ recognizing the language T. It is not hard to figure out a 1p2DFA recognizing $L \cdot T$ and employing $O(n+m)$ states. To achieve the upper bound one can use the pebble in the following way: suppose that the input is w, and suppose that the automaton has placed the pebble on cell i, then it checks if $w[1...i-1]$ belongs to L, and then if it is the case it checks if $w[i...|w|]$ belongs to T. If w pass both tests the automaton halts and accepts the input; otherwise it looks for the pebble, picks it up, places it on the next to the right cell and begins once again.

Thus, 1p2DFAs can real-state process the concatenation of two 2DFAs. Now, given $n \geq 1$, we set $T_n = L\left(a^{n-1}(a^n)^*\right)$. We know that T_n can be recognized employing a 2DFA with n states. Hence, given $n, m \geq 1$ the language $T_n \cdot T_m$ can be recognized by a 1p2DFA with $O(n+m)$ states. On the other hand, we know that if n and m are coprimes ($\gcd(n,m) = 1$) any 2DFA recognizing the language $(T_n \cdot T_m)$ requires $\Omega(nm)$ states. □

Remark 5.3 It is important to remark that a stronger separation result is already known. Let $\{p_i\}_{i\geq 1}$ be the enumeration of the prime numbers in ascending order. Given $m \geq 1$, we set $P_m = \prod_{i\leq m} p_i$, and we set $L_m = \left\{1^l : l < P_m\right\}$. Geffert and

Istonova [6] proved that the sequence $\{L_m\}_{m\geq 1}$ requires $\Omega\left(2^{m\log(m)}\right)$ states over the model of 2DFAs, but that it can be recognized using at most $O\left(m^2\log(m)\right)$ states over a model of 1p2DFAs studied by them. We have included the above proof because of three reasons: because our proof is a very simple proof which has been obtained thanks to our analysis of real-state conversion, (which seems to be a meaningful notion that will allow us to discover new proofs and new results), because their model of 1p2DFAs is stronger than our model (and then their proof could not hold in our case), and finally because we will get an interesting corollary (corollary 5.4) from the above proof:

Corollary 5.4 *1p2DFAs can real-state process the concatenation of two 2DFAs.*

The above corollary is not sufficient for our purposes, we have to ask: are 1p2DFAs able to real-state process concatenations of 1p2DFAs? Notice that if we try to use the naive idea used in the above proof, we will promptly realize that we need three pebbles. Are we allowed to use more than one pebble? It is not hard to figure out a two-pebble automaton recognizing the language of palindromes, and it is well known that this language is not regular. Thus, if we want to consider some type of automata using more than one pebble, we will have to impose some further constraints on the way those automata can handle their pebbles. We follow this direction of research in the next and last section, but before of this we would like to conclude with our analysis of concatenations over the model of 1p2DFAs.

Given n, we use the symbol H_n to denote the language $L\left((a^n)^*\right)$.

Lemma 5.5 *Let n, m be two integers, which are coprime, the number of states that are necessary to recognize the language $H_n\cdot H_m$ using 2DFAs is at least $nm-n+m-1$.*

Proof. Given an unary regular language L, and given $\mathcal{M}$, a minimal DFA accepting L, if the tail of $\mathcal{M}$ is equal to l, then a minimal 2DFA recognizing L has at least $l-1$ states. Notice that the language $H_n\cdot H_m$ is cofinite, and notice that the largest string that is not contained in this language is the string $a^{mn-(m+n)}$. Then, we have that the tail of a minimal DFA recognizing $H_n\cdot H_m$ is equal to $mn-(m+n)$, and then we have that any 2DFA recognizing this language has at least $nm-n+m-1$ states. □

Theorem 5.6 *Let n, m, s be three different prime numbers, we have that any 1p2DFA recognizing the language $H_n\cdot H_m\cdot H_s$ requires $\Omega\left(\min\{nm, ms, ns\}\right)$ states.*

Proof. Let $\mathcal{M}$ be a minimal 1p2DFA accepting the language $H_n\cdot H_m\cdot H_s$. Given $w\in\{a\}^*$, string w determines a unique computation of the automaton $\mathcal{M}$, which is divided in at most $|w|$ stages. If $w\in H_n\cdot H_m\cdot H_s$, it becomes accepted only because of the last stage of the computation, which begins when automaton places its pebble on a given cell, say i, dividing in this way the input string into two strings $w[1...i-1]$ and $w[i...|w|]$.

We define

$$L=\{w[1...i-1]: w\in\{a\}^*\ \&\ \alpha_{\mathcal{M}}(w,i)\}$$

and

$$T = \{w[i...|w|] : w \in \{a\}^* \ \& \ \beta_{\mathcal{M}}(w,i)\}$$

where $\alpha_{\mathcal{M}}(w,i)$ is supposed to mean:

If the computation of automaton $\mathcal{M}$, on input w, begins with the pebble placed on cell i, the workhead located on the left-end of the input and the internal state of $\mathcal{M}$ being equal to the L-initial state, then $\mathcal{M}$ will reach a transition-accepting state before picking up the pebble.

And $\beta_{\mathcal{M}}(w,i)$ is supposed to mean:

If the computation of automaton $\mathcal{M}$, on input w, begins with the pebble placed on cell i, the workhead located on cell i, and the internal state of $\mathcal{M}$ being equal to the R-initial state, then $\mathcal{M}$ reaches a transition-accepting state.

We have that $\mathcal{M}$ accepts w at the *ith* stage if and only if $w[1...i-1] \in L$, and $w[i...|w|] \in T$. Therefore, we have that $\mathcal{M}$ accepts w if and only if $w \in L \cdot T$, and it is equivalent to claim that $H_n \cdot H_m \cdot H_s = L \cdot T$. Notice that L and T are regular languages. One can use the primality of n, m and s to prove that there exists a regular language Ω, and that there exist $x, y \in \{n, m, s\}$ (with $x \neq y$) such that either $L = H_x \cdot H_y \cdot \Omega$ or $T = H_x \cdot H_y \cdot \Omega$. We can suppose, without loss of generality, that $L = H_x \cdot H_y \cdot \Omega$. Notice that one can use $\mathcal{M}$, without the pebble, to recognize the language L, it implies that the number of states of $\mathcal{M}$ is bigger than the minimum number of states that are necessary to recognize L using 2DFAs. It is easy to check that the state complexity of unary 2DFAs can only increase with concatenations, and then the number of states of automaton $\mathcal{M}$ is bigger than the number of states that are required to recognize the language $H_x \cdot H_y$ using 2DFAs. Thus, we have that $\mathcal{M}$ has at least $\Omega(xy)$ states. Therefore, we get the lower bound. □

Corollary 5.7 *1p2DFAs do not have real-state processing of concatenations.*

6 Deterministic Two–Way Multipebble Automata

So far, we have surveyed the most popular models of finite state automata, we showed that all those models, but the model of 1NFA, are unable of real-state processing regular expressions. It seems that real-state processing is not achievable within the world of deterministic finite state automata. Therefore we decided to relax our goal: we would be happy if we could find a model that is able of efficiently processing all the star free regular expressions (a model that is able of real-state processing unions and concatenations).

In this section we study the ultimate model of deterministic finite state automata, which is the model of *directed multipebble deterministic two-way finite automata* (dp2DFA, for short) introduced below. We prove that it is more powerful (succinct) than the model 1p2DFA, and we prove that this new model holds real-state processing of concatenations and unions (real-state processing of star free regular expressions).

A directed multipebble deterministic two-way finite automata is a pebble automaton provided with a fixed number of pebbles which can be larger than 1.

We mentioned before that one can construct a two pebble automaton recognizing palindromes, hence we have to impose some strong restrictions on the way those automata can handle their pebbles.

Suppose that we have a dp2DFA with k pebbles, say $P_1, ..., P_k$, which is processing the input string w, and suppose that it has placed its first i pebbles on cells $j_1, ..., j_i$. To begin, we demand that $j_1 \leq j_2 \leq ... \leq j_i$ and that the pebbles were placed one by one respecting the order of their labels (which are $1, ..., k-1$ and k). We suppose that the workhead is placed on the interval containing the string $w\left[j_{i-1}...j_i - 1\right]$. It must stay on this cell interval till it reaches a transition state. Depending on the transition state it reaches, it makes one out of two things: either the automaton looks for P_i, picks it up, places it on the next to the right cell and begins to work on the substring $w\left[j_{i-1}+1...j_i\right]$; or it looks for P_i, locates it, moves one step rightward, places P_{i+1} on cell $j_i + 1$ (thus, $j_i + 1 = j_{i+1}$) and begins to work on the string $w\left[j_i...j_{i+1} - 1\right]$.

It is not hard to formalize the definition of our dp2DFAs, nevertheless we will omit writing down this definition because it happens to be a little bit cumbersome. The key idea is that those automata used their k pebbles to partition their tapes into $k+1$ segments, and then they work on each one of those segments in an independent and sequential way. Moreover, they are designed to consider all the possible partitions of the input string into $k+1$ substrings. Thus, one could say that those automata are tailor-made to deal with concatenations. We will see that it is actually the case, we will see that those automata have real-state processing of concatenations, but before of this we have to check that this new class of automata accepts the regular languages.

Theorem 6.1 *dp2DFA accept the regular languages.*

Proof. dp2DFAs accept all the regular languages because a 2DFA is a dp2DFA that never uses its provision of pebbles. Now, we check that those two-way automata, provided with multiple pebbles, can only accept regular languages. Noa Globerman and David Harel studied in [7] a different model of multipebble automata which, they proved, accept the regular languages. A Globerman-Harel automaton is a two-way automaton with k pebbles ($k \geq 0$), say $P_1, \ldots, P_k$, that adheres to the following restrictions:

(i) P_{i+1} may not be placed unless P_i is already on the tape, and P_i may not be picked up unless P_{i+1} is not on the tape (Thus the pebbles are placed and picked up in a LIFO style).

(ii) Between the time P_{i+1} is placed and the time either P_i is picked up or P_{i+2} is placed, the automaton can traverse only the substring located between the current location of P_i and the end of the input word that lies in the direction of P_{i+1}. Moreover, in this substring, the automaton can act only as a 1p2DFA using P_{i+1} as its unique pebble. In particular, it is not allowed to lift up, place, or even sense the presence of any other pebble.

We notice that dp2DFAs adhere to the restrictions imposed on Globerman-Harel automata. Then, a dp2DFA automaton cannot recognize a nonregular language □

Can we exploit the multiple pebbles to get real-state processing of concatenations? Before studying any possible answer to this question a warning is in order: we are using a new computational resource, the pebbles, which must be quantified.

Definition 6.2 Let $\circ \in \{\cup, \cdot\}$, we say that dp2DFAs real-state process operation $\circ$ if and only if there exist a constant $C_\circ$ and a polynomial $p(X, Y)$ such that given two dp2DFAs, say $\mathcal{M}$ and $\mathcal{N}$, there exists a dp2DFA $\mathcal{K}$ recognizing the language $L(\mathcal{M}) \circ L(\mathcal{N})$ and such that:

(i) The number of states of $\mathcal{K}$ is bounded by $|Q_{\mathcal{M}}| + |Q_{\mathcal{N}}| + C_\circ$.

(ii) The number of pebbles of $\mathcal{K}$ is bounded by $p(\#\mathcal{M}, \#\mathcal{N})$, where $\#\mathcal{X}$ denotes the number of pebbles of automaton $\mathcal{X}$ ($\mathcal{X} \in \{\mathcal{M}, \mathcal{N}\}$).

Notice that dp2DFA can real-state process unions. Thus, we have to focus our attention on concatenations. We will prove that dp2DFAs are able of real-state processing concatenations. First a warm up.

Proposition 6.3 *Let m, n, s be three natural numbers, the language $H_m \cdot H_n \cdot H_s$ can be recognized employing a dp2DFA with two pebbles and $n + m + s + C$ states, where C is a constant that does not depend on the triple (n, m, s).*

Proof. First at all we recall that given $n \geq 1$, the language H_n can be recognized by a 2DFA with n states. Thus, we pick three 2DFAs $\mathcal{M}_m, \mathcal{M}_n$ and $\mathcal{M}_s$ recognizing the three languages H_m, H_n and H_s, and such that each one of those three automata has n, m and s states (respectively). Now we construct a dp2DFA $\mathcal{N}$ with two pebbles P_1 and P_2, and which merges together the three automata introduced before. Automaton $\mathcal{N}$ works, on input w, as follows:

Suppose that $\mathcal{N}$ has placed P_1 on cell i, then it checks if the string $w[1...i-1]$ belongs to H_m. If it is not the case it enters a transition state, looks for the pebble, enters the right side, picks up the pebble, places it on the next to right cell and begins once again. Otherwise (i.e. if $w[1...i-1]$ belongs to H_m), automaton $\mathcal{N}$ enters a second different transition state and looks for the right portion of the input string. Notice that, from the exact moment $\mathcal{N}$ enters the right portion of the tape till it picks up the pebble P_1 once again, it is forced to work on the substring $w[i...|w|]$. Along this period of time automaton $\mathcal{N}$ uses P_2 to simulate the pair $\mathcal{M}_n$ and $\mathcal{M}_s$, while checking if $w[i...|w|]$ belongs to $H_n \cdot H_s$. If $w[i...|w|] \in H_n \cdot H_s$, automaton $\mathcal{N}$ halts and accepts the input, otherwise it picks up pebble P_2, picks up pebble P_1, advances one step to the right, places P_1 on this cell and begins once again. □

We get from the above proposition an interesting corollary

Corollary 6.4 *dp2DFA cannot be linearly simulated by 1p2DFA, even when restricted to the unary case.*

Remark 6.5 We can elaborate on the proof idea used in theorem 5.6, to get the following more general result: Let $k \geq 2$, and let $m_1, ..., m_k$ be k different prime numbers such that no one of them is a positive integer combination of the others, the language $H_{m_1} \cdot ... \cdot H_{m_k}$ can be recognized using an automaton with $m_1 + ... + m_k$ states and $k-1$ pebbles, while any dp2DFA with $k-2$ pebbles requires $\Omega\left(\min\left\{m_i m_j : i, j \leq k \text{ and } i \neq j\right\}\right)$ states. It implies that for all $k \geq 2$, directed pebble automata with k pebbles cannot be linearly simulated by directed pebble automata with $k-1$ pebbles. It implies that each additional pebble can represent an important gain in computing power. It is important to stress that Globermann and Harel proved a similar result for its model of multipebble automata [7].

Theorem 6.6 *dp2DFAs are able of real-state processing concatenations.*

Proof. We only have to elaborate on the proof idea that was used in proposition 6.3. The rough idea is the following one: suppose that we have two dp2DFA, say $\mathcal{M}_1$ and $\mathcal{M}_2$, each with n_i states and k_i pebbles $(i = 1, 2)$. We define a new dp2DFA denoted with the symbol $\mathcal{N}$. Automaton $\mathcal{N}$ has $n_1 + n_2 + C$ states, $k_1 + k_2 + 1$ pebbles and works as follows: Suppose that it has detected that the prefix $w\left[1...i-1\right]$ belongs to $L\left(\mathcal{M}_1\right)$, suppose that it has placed the first $k_1 + 1$ pebbles on the tape and suppose that the last one is placed on cell i. From this exact moment till the moment P_{k_1+1} is picked up again, it works on the suffix $w\left[i...\left|w\right|\right]$ while simulating the automaton $\mathcal{M}_2$ with the help of the remaining k_2 pebbles □

It seems that dpDFA cannot real-state process the Kleene star. If we try to use the naive idea employed in the case of concatenations we will promptly realize that we have to use an unbounded number of pebbles. An unbounded number of pebbles seems to be a not admissible resource because , among other things, we need to include some special states in order to handle the provision of pebbles, and it happens that the number of those states increases with the number of pebbles. Thus, such a model of pebble automata seems to be nonfeasible (seems to be nonfinite).

We conjecture that there does not exist a feasible deterministic model of finite automata for which Thompson property holds. Our conjecture implies that there does not exist a feasible deterministic model of finite automata that is able of real-state processing the regular operations. We have that dp2DFAs are able of real-state processing the star free regular expressions and it is the best result that we can achieve so far. Thus, we have

Proposition 6.7 *There exists an algorithm which, on input α (where α is a star free regular expression), computes in linear time a dp2DFA with $O\left(\left|\alpha\right|\right)$ states and $O\left(\left|\alpha\right|\right)$ pebbles that recognizes the language $L\left(\alpha\right)$.*

There is a third computational resource employed by dp2DFA which must be quantified: running time. Two-way automata can work under different running time regimes, and the running time of a given two-way automaton cannot be bounded apriory. Thus, it is natural to ask about the running times of the automata that can be obtained as outputs of the algorithm mentioned in the statement of proposition 6.7. It is not hard to check that given α, a regular expression, the running time of $\mathcal{M}_\alpha$, which is the dp2DFA computed by the aforementioned algorithm, belongs to $O\left(n^{\left|\alpha\right|}\right)$. Moreover, it can be proved that given k, there exists α such that the running time of $\mathcal{M}_\alpha$ belongs to $\Omega\left(n^k\right)$.

Thus, we can conclude that the model of dp2DFAs does not behave well when it comes to the analysis of running time. We conclude with a conjecture

Conjecture 6.8 *There does not exist a class of linear time finite state deterministic automata having real-state processing of star free regular expressions.*

7 Conclusion

One can argue that The Sakoda-Sipser Problem is the question about the state-complexity of simulating a given class of finite automata by another one. Thus, from this very general point of view, The Sakoda-Sipser problem is a question about comparing the state-complexity of different computational tasks when they are analyzed through the lenses of different models of finite automata. We chosen one specific task: processing of regular expressions. Our choice yields a new and meaningful notion: Real-state conversion. The analysis of this new notion allowed us to explain, to some extent, what is special about 1NFAs, and which are the main computational advantages of nondeterminism when one restricts the attention to finite automata. We could prove some preliminary results concerning this new notion, but we feel that it deserves further investigation.

References

[1] Birget, J. *Intersection and union of regular languages and state complexity*, Information Processing Letter, **43** (1992), 185–190.

[2] Chang J., Ibarra O., Palis M., and Ravikunar B., *On pebble automata*, Theoretical Computer Science, **44** (1986), 111–121.

[3] Chrobak M., *Finite automata and unary languages*, Theoretical Computer Science, **47** (1986), 149–158.

[4] Ehrenfeucht A. and Zeiger P., *Complexity measures for regular expressions*, Journal of Computer and System Sciences, **12** (1976), 134–146.

[5] Geffert V., Mereghetti C., and Pighizzini G., *Complementing two-way finite languages*, Information and Computation, **205** (2007), 1173–1187.

[6] Geffert V. and Istonova L., Translation from classical two-way automata to pebble two-way automata, in 11th International Workshop on Descriptional Complexity of Formal Systems (DCFS 2009), Magdeburg, Germany, Jul. 2009, 131–140.

[7] Globerman N. and Harel D., *Complexity for two way and multi-pebble automata and their logics*, Theoretical Computer Science, **169** (1996), 161–184.

[8] Sakoda W. and Sipser M., Nondeterminism and the size of two way finite automata, in STOC 78 Proceedings of the tenth annual ACM symposium on Theory of computing, 1978, 275–286.

[9] Shallit J., "A second course in Formal Languages and Automata Theory". Cambridge, MA: Cambridge University Press, 2009.

[10] Thompson K., *Programming techniques: Regular expression search algorithm*, Communications of the ACM, **11** (1968), 419–422.

[11] Yu S., Zhuang Q., and Salomaa K., *The state complexities of some basic operations on regular languages*, Theoretical Computer Science, **125** (1994), 315–328.