Block-Deterministic Regular Languages*

Dora Giammarresi[†] Rosa Montalbano[‡] Derick Wood[§]

September 11, 1998

Abstract

We introduce the notions of blocked, block-marked and block-deterministic regular expressions. We characterize block-deterministic regular expressions with deterministic Glushkov block automata. The results can be viewed as a generalization of the characterization of one-unambiguous regular expressions with deterministic Glushkov automata. In addition, when a language L has a block-deterministic expression E, we can construct a deterministic finite-state automaton for L that has size linear in the size of E.

1 Introduction

A regular language is one-unambiguous, according to Brüggemann-Klein and Wood [4], if there is a deterministic Glushkov automaton for the language. An alternative definition of one-unambiguity based on regular expressions is that each position in a regular expression has at most one following position for each symbol in the expression's alphabet. The latter definition is used to define unambiguous content model groups in the Standard Generalized Markup Language (SGML) [11], which are a variant of regular expressions. Indeed, it was the SGML standard that motivated Brüggemann-Klein and Wood's investigation of one-unambiguity. In contrast, to the results of Book and his coworkers [3] on ambiguity of regular expressions, there are regular languages that are not one-unambiguous [4]. It is clear, from the definition of one-unambiguity, that when a regular expression is one-unambiguous it is also unambiguous in the sense of Book and his colleagues. The difference is that one-unambiguity can also be viewed as one-determinism. A lookahead of one symbol when processing a string from left to right determines a unique next position in the given regular expression; they are, essentially, LL(1) regular expressions [1, 4].

These observations lead to two possible generalizations (at least) of one-unambiguous regular expressions. The first is based on a lookahead of at most $k \geq 1$ symbols to determine the next, at most one, matching position in a regular expression. The second

*This research was partially supported under a grant from the Research Grants Council of Hong Kong SAR.

[†]Dipartimento di Matematica applicata e Informatica, Università Ca' Foscari di Venezia, via Torino 155, 30173 Venezia Mestre, Italy. Email: dora@dsi.unive.it

[‡]Dipartimento di Matematica e Applicazioni, Università di Palermo, via Archirafi 34, 90123 Palermo, Italy. Email: rosalba@altair.math.unipa.it

[§]Department of Computer Science. Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong SAR. Email: dwood@cs.ust.hk

is similar except that when we use a lookahead of l symbols we must match the next l positions uniquely. The first notion defines k-unambiguous expressions and the second defines k-block-deterministic expressions. We focus on k-block-deterministic expressions in this work.

Our results have an interesting implication about the regular languages that have block-deterministic expressions. When a language L has a k-block-deterministic expression E, we can construct a deterministic finite-state automaton for L that has size linear in the size of E. Are there other "natural" classes of regular expressions that have this property?

In Section 2, we review basic notation and terminology and, in Section 3, we introduce blocked expressions, block-marked expressions and block-deterministic expressions. In Section 4, we characterize block-deterministic languages in terms of block-deterministic automata.

2 Notation and terminology

Let Σ be an alphabet of symbols. Regular expressions over Σ are built from λ, $\emptyset$, and symbols in Σ using the binary operators $+$ and $\cdot$ and the unary operator *. The language specified by a regular expression E is denoted by $\mathrm{L}(E)$. The symbols that occur in a regular expression E are denoted by $\mathrm{sym}(E)$.

To indicate different positions of the same symbol in a regular expression, we mark symbols with unique subscripts. For example, $(a_1+b_1)^*a_2(a_3b_2)^*$ and $(a_4+b_2)^*a_1(a_5b_1)^*$ are both **markings** of the regular expression $(a+b)^*a(ab)^*$. For each regular expression E over Σ, a marking of E is denoted by E'. If H is a subexpression of E, we assume that markings H' and E' are chosen in such a way that H' is a subexpression of E'. A **marked regular expression** E' is a regular expression over Π, the alphabet of subscripted symbols, where each subscripted symbol occurs at most once in E'.

The reverse of marking is the dropping of subscripts, indicated by $^\natural$ and defined as follows: If E is a regular expression over Π, then $E^\natural$ is the regular expression over Σ that is obtained from E by dropping all subscripts in E. Thus, a marked regular expression H is a **marking** of regular expression E if and only if $H^\natural = E$. Unmarking can also be extended to words and languages: For a word w over Π, let $w^\natural$ denote the word over Σ that is constructed from w by dropping all subscripts. For a language L over Π, let $L^\natural$ denote $\{w^\natural | w \in L\}$. Then, for each regular expression E over Π, $\mathrm{L}(E^\natural) = \mathrm{L}(E)^\natural$.

Book and his associates [3] and Eilenberg [6] define ambiguous regular expressions as follows. A regular expression E is **ambiguous,** if, for a marked version E' of E and for some $w \in L(E)$, there are two marked words $x, y \in L(E')$ such that $x \neq y$ and $x^\natural = y^\natural = w$. An expression E is **unambiguous** if it is not ambiguous. A regular language L is **unambiguous** if it is denoted by an unambiguous regular expression. Book and his associates [3] proved that all regular languages are unambiguous. Brüggemann-Klein and Wood [4] defined a more restrictive version of unambiguity motivated by SGML content models [11]. A regular expression E is **one-unambiguous** if and only if, for all words u, v and w over Π, the alphabet of subscripted symbols, and all symbols x and y in Π, the conditions $uxv, uyw \in L(E')$ and $x \neq y$ imply that that $x^\natural \neq y^\natural$. A regular language is one-unambiguous if it is denoted by some one-unambiguous expression.

Given a regular expression E, we can construct an automaton that recognizes $L(E)$

in many different ways. Many of these automata can be reduced to the **Glushkov automaton** [4, 8]. Glushkov first suggested this construction in 1960 [9, 10]; it was also suggested by McNaughton and Yamada [12] independently and at about the same time. The construction is based on the first, last and follow sets of positions in the given regular expression. We define the three sets of positions as follows:

$\text{first}(E')$ is the set of all positions that can begin a string in $L(E')$.

$\text{last}(E')$ is the set of all positions that can end a string in $L(E')$.

$\text{follow}(a, E')$ is the set of all positions in $L(E')$ that can follow position a.

Once we have computed these sets, we can construct the Glushkov automaton G_E directly as follows: *The states of* G_E *are* $\Pi \cup \{0\}$, 0 *is the start state,* $\text{last}(E')$ *is the set of final states, and the transitions are*

$$\{(x, a, a_j) : a_j \in \text{follow}(x, E') \textit{ or } x = 0 \textit{ and } a_j \in \text{first}(E')\}.$$

Caron and Ziadi [5] recently characterized Glushkov automata.

Block automata[1] were introduced by Eilenberg [6]. They allow the transition labels to be nonempty strings or **blocks** over the input alphabet rather than just symbols. Formally, a *block automaton* A is specified by a tuple $(Q, \Sigma, \Gamma, E, s, F)$, where Q is a finite set of states, Σ is an input alphabet, Γ is a finite subset of Σ^+ called the **block alphabet,** $\delta \subseteq Q \times \Gamma \times Q$ is a transition relation, $s \in Q$ is a start state and $F \subseteq Q$ is a set of final states.

As with standard finite-state automata we define a string x to have an accepting computation in a block automaton A if there is a path from the start state to some final state that spells the string x. If there is more than one such path, the automaton is nondeterministic and, therefore, ambiguous. The collection of all strings that have an accepting computation in a block automaton A is called the **language of A** and we denote it by $L(A)$. In finite-state automata, if a string has more than one accepting computation, then not only is the automaton nondeterministic, but also it implies there is a state that has two outgoing transitions with the same label. This implication does not necessarily hold for block automata; we have the following weaker implication. Nondeterminism occurs in a block automaton when there is a state that has two outgoing transitions such that the label of one transition is a prefix of the label of the other transition. The absence of nondeterminism in a block automaton corresponds to the set of all blocks in transitions from a given state being prefix free. Deterministic block automata were introduced by Giammarresi and Montalbano [7] when they investigated the minimization of block automata. We use deterministic block automata to define block-deterministic regular languages.

More formally, let $A = (Q, \Sigma, \Gamma, \delta, s, F)$ be a block automaton. Now, for each $q \in Q$, let $\text{block}(q) \subseteq \Gamma$ be the set of blocks in the transitions out of q. Then, A is a **deterministic block automaton** if, for every state $q \in Q$, $\text{block}(q)$ is prefix-free.

Given a block automaton, if the maximum block length is k, then we refer to it as a **k-block automaton.** From this viewpoint, a standard finite-state automaton is a one-block automaton. Block automata are, therefore, a generalization of standard finite-state

[1] Block automata are called generalized automata by Eilenberg [6].

automata that also describe all and only regular languages. Moreover, observe that block automata are standard finite-state automata when we treat the blocks in the transitions as labels—as we do whenever we refer to the elements of a block alphabet. With this assumption, we can apply the usual automata transformations, such as state minimization and determinization, to block automata. Given a block automaton A, we denote its minimal and deterministic automata by $\mathcal{M}(A)$ and $\mathcal{D}(A)$, respectively, when considering its blocks as labels.

We now describe two transformations that are essentially mutual inverses of each other: **state elimination** and **block expansion.**

Let A be a block automaton and q be a state of A such that q is not the start state, it is not a final state and it has no self-loops. We define the **state elimination of q in A** as follows: We first remove state q and all transitions into and out of q from A. Second, for every pair (r, u, q) and (q, v, s) of transitions that were in A, we add a new transition (r, uv, s) to A. We denote the resulting automaton by $\mathcal{S}(A, q)$. It is easy to verify that $\mathcal{S}(A, q)$ is indeed a block automaton equivalent to A. We can also extend state elimination to a set $S \subset Q$ of states. Giammarresi and Montalbano [7] prove that if S does not contain the start state and any final state, and the subgraph induced by S is acyclic, then we can construct a unique block automaton $\mathcal{S}(A, S)$ by eliminating the states in S in any order. In this case we say that the set $S \subseteq Q$ of states satisfies the **state-elimination precondition** for A. Notice that when S induces an acyclic subgraph of A and A is a k-block automaton, the length of blocks in $\mathcal{S}(A, S)$ can increase to at most $|S|k$. Finally, if we apply state elimination to any state of a deterministic finite-state automaton that satisfies the precondition, then we obtain a deterministic block automaton. Note, however, that we can also obtain deterministic block automata from nondeterministic finite-state automata.

The second transformation, block expansion, is defined on the transitions of an automaton. It takes a transition and expands it into a sequence of transitions with single-symbol labels. More precisely, given a transition $e = (p, a_1 a_2 \cdots a_k, q)$ in A, where $k \geq 2$, we define the **block expansion of e in A** as follows: We remove the transition e from A and introduce new states $p_1, \ldots, p_{k-1}$ and new transitions $(p, a_1, p_1), (p_1, a_2, p_2), \ldots, (p_{k-1}, a_k, q)$. We denote the resulting block automaton by $\mathcal{B}(A, e)$. Clearly, given a block automaton A, we can expand it to give a finite-state automaton by applying block expansion to all appropriate transitions of A. Thus, when A is a deterministic block automaton, the resulting finite-state automaton need not be deterministic.

3 Block-deterministic regular expressions

We define block-marked regular expressions, and block-deterministic regular expressions and languages in Section 3.1. Then, in Section 3.2, we characterize block-deterministic regular languages as those languages defined by deterministic block automata.

3.1 Block marking and Glushkov block automata

Let E be a regular expression over an alphabet Σ. We define a **block** of E to be a dotted subexpression of E. For example, given the expression

$$E = (a \cdot a)^* \cdot (a \cdot b \cdot b + b \cdot a) \cdot b^*,$$

then a, aa, ab, abb, b, ba and bb are all possible blocks in E, whereas aab and bbb are not blocks of E although they are substrings of strings in $L(E)$. We can **partition** the dotted expressions in a regular expression E into blocks such that each appearance of a symbol appears in exactly one block. We can partition the running-example expression in more than two ways; for example, we obtain six blocks with the partition

$$([a][a])^*([ab][b] + [ba])([b])^*,$$

where we use square brackets [and] to enclose blocks. There is the minimum partition of a regular expression that treats each maximal dotted subexpression as a block; for example,

$$([aa])^*([abb] + [ba])([b])^*$$

has four blocks. There is also the maximum partition that treats each single symbol as a block; for example,

$$([a][a])^*([a][b][b] + [b][a])([b])^*,$$

has eight blocks. An expression that is partioned into blocks is called a **blocked expression.**

We define a **block marking** of an expression using a blocked version of the expression. A *block marking* of an expression E is obtained by partitioning E into blocks and uniquely marking each block with an integer subscript. When we wish to identify the maximum length, k, of the blocks in a block marking, we call it a **k-block marking.** We denote a block-marked version of an expression E by E'. We denote by block(E') the set of all marked blocks of E'. Thus, a block-marked regular expression E' is a regular expression over the alphabet $\Gamma = \text{block}(E')$. For example, one block marking for the running example E is

$$E' = ([a]_1[a]_2)^*([ab]_3[b]_4 + [ba]_5)([b]_6)^*,$$

in which case

$$\text{block}(E') = \{[a]_1, [a]_2, [ab]_3, [b]_4, [ba]_5, [b]_6\}.$$

The **unmarking** of a block-marked expression removes all subscripts and the square brackets. If E is a blocked-marked expression, then $E^\natural$ is the corresponding unmarked expression.

Block marking and unmarking of regular expressions can be extended in an obvious way to block marking and unmarking of words and languages. Notice that block marking generalizes the notion of marked expressions [9, 12, 2, 4] that corresponds to all blocks being of length one. Now, given a block-marked regular expression E', we can extend to E' the functions first, last and follow introduced by Glushkov, McNaughton and Yamada [9, 12]. In this case, $\text{first}(E')$, $\text{last}(E')$ and $\text{follow}(E', x)$ are subsets of $\text{block}(E') = \Gamma$. Using these sets, we give a formal definition of block-deterministic regular expressions.

A regular expression E is **block deterministic** if and only if there is a block marking E' of E such that the following two conditions hold:

1. For all $x, y \in \text{first}(E')$, $x \neq y$ implies that $x^\natural$ is not a prefix of $y^\natural$.
2. For all $z \in \text{block}(E')$ and for all $x, y \in \text{follow}(E', z)$, $x \neq y$ implies that $x^\natural$ is not a prefix of $y^\natural$.

A block marking that satisfies the two preceding conditions is called a **deterministic block marking.** If we restrict the block length to one, then one-block-deterministic expressions coincide with one-unambiguous expressions as defined by Brüggemann-Klein and Wood [4]. In general, a deterministic block marking for a given block-deterministic regular expression E is not unique. This observation holds even when the maximal length k of the blocks is specified. As an example, consider the running example expression $E = (aa)^*(abb + ba)(b)^*$. There are two different deterministic two-block markings for E:

$$E_1' = ([aa]_1)^*([ab]_2[b]_3 + [b]_4[a]_5)([b]_6)^*$$

and

$$E_2' = ([aa]_1)^*([ab]_2[b]_3 + [ba]_4)([b]_5)^*.$$

We will follow the following conventions throughout. First, when we refer to Γ, we mean the set of labels that are treated as atomic symbols and when we refer to $\text{block}(E')$, we mean the set of blocks that are treated as strings. Second, when we refer to finite-state automata we mean the standard model.

Given a block-marked expression E', the Glushkov automaton for the corresponding block set $\text{block}(E')$ is called the **Glushkov block automaton** and we denote it by $G^k(E')$, where E' is a k-block marking of E. Observe that the Glushkov block automaton for a given regular expression E depends on the block marking of E. We use Glushkov block automata to characterize block-deterministic regular expressions. This characterization generalizes the one of Brüggemann-Klein and Wood [4] to the case of k-block markings for $k > 1$.

The proof of the following result is a direct consequence of the definitions of deterministic block automata and of block-deterministic regular expressions.

Lemma 3.1 *A regular expression E is block deterministic if and only if there is a k-block marking E' of E such that the Glushkov block automaton $G^k(E')$ is deterministic.*

If we want to emphasize the maximal length k of the blocks, we write **k-block deterministic.** Equivalently, a k-block marking E' for E is deterministic if and only if the corresponding Glushkov block automaton $G^k(E')$ is deterministic.

Instead of introducing block automata, we could also consider the conventional Glushkov finite-state automaton of the expression defined on the alphabet Γ. Unfortunately, the intuitive characterization "A k-block marking of E is deterministic if and only if some Glushkov automaton for the corresponding block alphabet is deterministic" does not hold. We need the stronger condition that, for each state q in the Glushkov automaton, the blocks of the transitions out of q form a prefix-free set.

3.2 A characterization theorem

We now consider the problem of deciding whether a given regular expression E is block-deterministic. One simple method to solve this problem is to guess a $k \geq 1$ and a k-block marking E' and then construct the corresponding Glushkov k-block automaton $G^k(E')$. If $G^k(E')$ is deterministic, then E is k-block-deterministic.

Note that, for the case $k = 1$, the problem is easy to solve since there is a unique one-block marking and the corresponding Glushkov block automaton is the Glushkov finite-state automaton G_E. In this case, if G_E is deterministic, then E is one-deterministic. We now consider the case when G_E is not deterministic and describe a procedure to determine whether there is a deterministic k-block marking for E, for some $k \geq 2$. More precisely, such a k-block marking will be one with the minimum k.

Lemma 3.2 *If a regular expression E is k-block-deterministic, then its corresponding Glushkov finite-state automaton G_E can be transformed into a deterministic block automaton by a sequence of state eliminations.*

Proof: Let E' be a deterministic k-block marking for E and let $G^k(E')$ be the corresponding (deterministic) Glushkov block automaton. We apply to $G^k(E')$ a sequence of block expansions (see Section 2 for the definition) to all appropriate transitions.In this way, we transform $G^k(E')$ into a finite-state automaton $G^\natural$. Notice that, to reconstruct the block automaton $G^k(E')$, we simply apply state eliminations of all the new states. Moreover, it is easy to verify that $G^\natural = G_E$. Indeed, all the states introduced by block expansions correspond to the positions of the symbols in the blocks of E' that are not the last symbols of the blocks. $\square$

As a consequence of Lemma 3.2, given the Glushkov automaton G_E of a k-block-deterministic expression E, all the states that are responsible for its nondeterminism can be eliminated to give a deterministic block automaton. In this case, a finite sequence of state eliminations can be used instead of subset construction to determinize G_E. We obtain a deterministic block automaton, but it can be considered to be a finite-state automaton defined on the block alphabet.

We now identify the responsible states. Let A be a (block) automaton and let q_1 and q_2 be two different states of A. Then, q_1 and q_2 are **duplicates,** if one of the following two conditions holds:

1. For some state p and some $x \in \Sigma$ ($x \in \Sigma^*$), both transitions (p, x, q_1) and (p, x, q_2) are in A.

2. For two duplicate states p_1 and p_2 and some $x \in \Sigma$ ($x \in \Sigma^*$), both transitions (p_1, x, q_1) and (p_2, x, q_2) are in A.

Recall that, given an automaton A, if we apply the subset construction to A we get a deterministic automaton $\mathcal{D}(A)$ whose states are subsets of the original set of states of A. We refer to a state of $\mathcal{D}(A)$ as either a **multiple state** or as a **single state** according to the cardinality of such sets. A state q of A is possibly included in several states, single and multiple, of $\mathcal{D}(A)$.

It follows that the duplicate states of a given automaton A are those that are in multiple states in $\mathcal{D}(A)$. Therefore, an automaton is deterministic if and only if it does not have any duplicate states.

We state the following result without proof.

Lemma 3.3 *Let E be a regular expression and let G_E be the corresponding Glushkov automaton. If G_E can be transformed into a Glushkov block automaton by applying state elimination to all its duplicate states, then E is block deterministic.*

Lemmas 3.2 and 3.3 suggest the following theorem.

Theorem 3.1 *Let E be a regular expression and let G_E be the corresponding Glushkov automaton. Then, E is k-block deterministic, for some k, if and only if G_E can be transformed into a k-deterministic Glushkov automaton by eliminating all of its duplicate states. Moreover, this Glushkov automaton defines a deterministic k-block marking of E.*

In the sequel, if E is a k-block-deterministic regular expression, we denote by G_E^k the deterministic Glushkov k-block automaton obtained by applying state elimination to all duplicate states in G_E. Moreover, we will refer to the block marking induced by G_E^k as the **standard block marking** of E.

From the proof of the Lemma 3.3, we obtain the following algorithm to determine whether a given regular expression E is block deterministic. First compute the Glushkov automaton G_E and identify the set Q_{dup} of its duplicate states. If Q_{dup} satisfies the state-elimination precondition (it does not contain the start state or a final state and it induces an acyclic subgraph), then compute $G'_E = \mathcal{S}(G_E, Q_{dup})$. Second, determine whether G'_E is a Glushkov automaton for the block alphabet using, for example, the characterization of Caron and Ziadi [5]. If it is, then $G'_E = G_E^k$ defines a deterministic block marking of E.

Consider the running example expression E and its Glushkov automaton in Fig. 1(a). It contains two duplicate states; that is, the states in $Q_{dup} = \{1, 3\}$ satisfy the state-elimination precondition. By eliminating these states we obtain $G^k(E')$ that is a deterministic Glushkov automaton for the alphabet $\{a, aa, ab, b\}$.

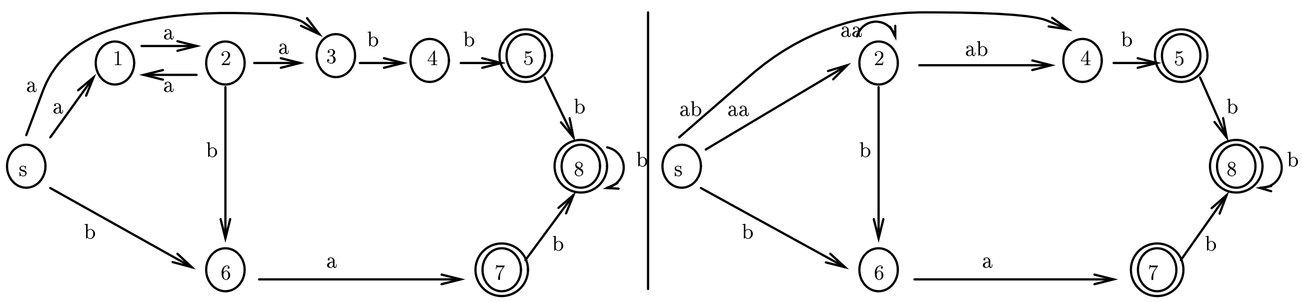


Figure 1: Two Glushkov automata for the running example expression E. a. The Glushkov automaton G_E for E. b. The deterministic Glushkov block automaton $G^2(E')$ for a two-block marking of E obtained by state elimination in G_E.

We conclude this section by mentioning that the application of subset construction to the Glushkov automaton G_E of a block-deterministic regular expression E does not increase the size of the automaton whereas, in the worst case, subset construction produces exponential blow-up. Indeed, from the proof of Lemma 3.3 we infer that the number of states of $\mathcal{D}(G_E)$ is at most the number of states of G_E since the set of duplicate states does not induce cycles.

4 Block-deterministic regular languages

A regular language L is **block deterministic** if there is a block-deterministic regular expression E such that $L = L(E)$. We now demonstrate that there are regular languages that are not block deterministic.

We first consider the problem of deciding whether a given regular language is block deterministic. The basic idea is to use the characterization established by Brüggemann-Klein and Wood [4] for unambiguous regular languages (one-block-deterministic regular languages in our terminology). Now, a regular expression is one-unambiguous if and only if its Glushkov automaton is deterministic. Brüggemann-Klein and Wood show that if a Glushkov automaton is deterministic, then it has some properties that are preserved under minimization. Therefore, such properties can be checked on the minimal finite-state automaton M for the given language. Moreover, if these properties hold for some minimal finite-state automaton, they prove that the corresponding regular language is one-unambiguous. Thus, they are able to give an algorithm that determines whether a given language is one-block deterministic and, if it is, they are able to construct a one-block-deterministic expression for it. We refer to this characterization as the **BW test for one-block-deterministic languages.** Suppose we want to test whether a given language $L \subset \Sigma^*$ is k-block deterministic for some fixed k. Let M be the minimal finite-state automaton for L. We apply state elimination to M to get a k-block automaton N^k. Let N be the same automaton as N^k considered as a minimal finite-state automaton on its block alphabet Γ. We can then apply the BW test to N. If L, considered to be over Γ, is one-block-deterministic, then there is a deterministic Glushkov automaton on Γ that reduces to N under minimization. Such a Glushkov automaton gives a k-block-deterministic regular expression together with a deterministic k-block marking for the original L ($L \subseteq \Sigma^*$).

On the other hand, if we consider all possible k-block automata that we can get from M by state elimination and none of them pass the BW test (when considered on the corresponding block alphabet), then we can conclude that L is not k-block deterministic for any k. This procedure always terminates. Given an automaton A, the number of all possible block automata obtained from A by state elimination is finite.

Notice that the preceding algorithm works only under the assumption that, given a block alphabet Γ, the minimal automaton N for L, when considered to be over Γ, can be obtained by applying state elimination to the minimal finite-state automaton M (the minimal automaton for L when considered to be over Σ). We show that this assumption is valid. If q is a state of a given automaton A, we let L_q denote the language recognized by A using q as the start state. The proof of the following result will be given in the full version.

Lemma 4.1 *Let L be a block-deterministic regular language. Then, there is a block-deterministic regular expression $E^\natural$ with the property that if p and q are two states of $\mathcal{D}(G_{E^\natural})$, then $L_p = L_q$ implies that either p and q are both sets of duplicate states of $G_{E^\natural}$ or p and q are both (single) non-duplicate states of $G_{E^\natural}$.*

Given a k-deterministic regular expression E, we let G_E and G_E^k be its corresponding Glushkov and k-block Glushkov automata, respectively. We consider the following two automata

$$M = \mathcal{M}(\mathcal{D}(G_E))$$

and

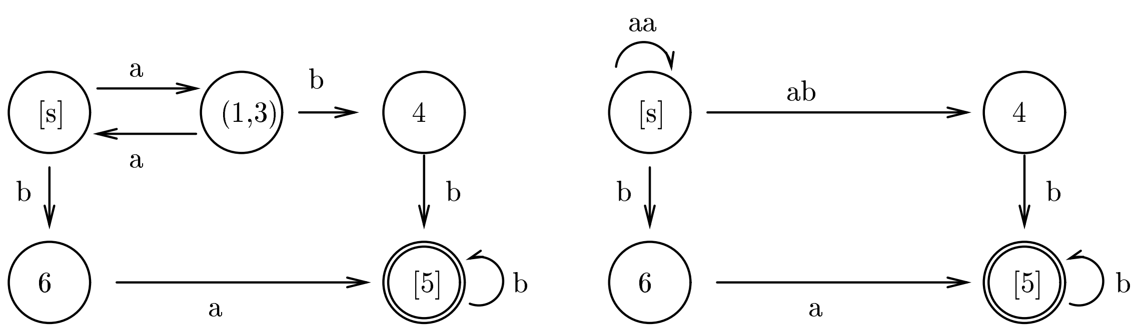
$$M^k = \mathcal{M}(G_E^k) = \mathcal{M}(\mathcal{S}(G_E, Q_{dup})),$$

where M is obtained from G_E by applying first subset construction and then minimization whereas M^k is obtained from G_E^k by applying minimization. (Equivalently, M^k is obtained from G_E by first applying state elimination of all duplicate states and then applying minimization.)

Lemma 4.2 *Let L be a k-deterministic language. Then, there is a block-deterministic regular expression E for L such that M can be transformed into M^k by state elimination.*

Proof: Let Q_M and Q_{M^k} be the sets of states of the automata M and M_k, respectively. By Lemma 4.1, Q_{M^k} is a proper subset of Q_M (or, more precisely, Q_M contains an isomorphic copy of Q_{M^k}). Moreover, all the states in $Q_M \setminus Q_{M^k}$ are classes of duplicate states of G_E and their corresponding transitions define an acyclic subgraph of M (the set of all such states satisfies the state-elimination precondition). □

Let us consider once again the running example expression E; that is, consider the language $L = L(E)$ on the alphabet $\Sigma = \{a, b\}$. The minimal finite-state automaton M for L in Fig. 2(a) is obtained by determinizing G_E of Fig. 1(a) and then minimizing it. When we apply the BW test to M, we see that L is not one-block deterministic. We then eliminate state $(1, 3)$ from M and obtain the automaton N_k of Fig. 2(b).



$M = \mathcal{M}(\mathcal{D}(G_E))$ $N_k = \mathcal{S}(M, \{1, 3\})$

Figure 2: A minimal finite-state automaton and state elimination. a. The minimal finite-state automaton M for the running example expression E. b. The result of eliminating state $(1, 3)$ in M.

N_k, considered as an automaton on the block alphabet $\Gamma = \{a, b, aa, ab\}$, can be obtained minimizing the deterministic Glushkov block automaton of Fig. 1(b), where states s and 2 are equivalent, and states 5, 7 and 8 are equivalent. These observations imply that L is a one-block-deterministic automaton on Γ and a two-block-deterministic automaton on Σ.

Using the same approach, we can exhibit languages that are not k-block deterministic, for any k; therefore, they are not k-deterministic. One example language is $L = \{a + b\}^*\{a\{a + b\}^n\}$. Brüggemann-Klein and Wood [4] prove that L is not one-block deterministic. Moreover, we can verify that it does not pass the BW test after the state elimination of all states that satisfy the state-elimination precondition.

References

[1] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation, and Compiling, Vol. I: Parsing.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1972.

[2] G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48:117–126, 1986.

[3] R.V. Book, S. Even, S.A. Greibach, and G. Ott. Ambiguity in graphs and expressions. *IEEE Transactions on Electronic Computers*, C-20:149–153, 1971.

[4] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 140:229–253, 1998.

[5] P. Caron and D. Ziadi. Characterization of Glushkov automata. *Theoretical Computer Science*, 1998. To appear.

[6] S. Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, New York, NY, 1974.

[7] D. Giammarresi and R. Montalbano. Deterministic generalized automata. *Theoretical Computer Science*, 1998. To appear. A preliminary version appeared in STACS '95, in *Springer-Verlag Lecture Notes in Computer Science 900*, 1995: 325–336.

[8] D. Giammarresi, J.-L Ponty, and D. Wood. The Glushkov and Thompson constructions: A synthesis. Unpublished manuscript, July 1998.

[9] V. M. Glushkov. On a synthesis algorithm for abstract automata. *Ukr. Matem. Zhurnal*, 12(2):147–156, 1960. In Russian.

[10] V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1961.

[11] ISO 8879: Information processing—Text and office systems—Standard Generalized Markup Language (SGML), October 1986. International Organization for Standardization.

[12] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9:39–47, 1960.