An improved construction of deterministic omega-automaton using derivatives*

Roman R. Redziejowski

Abstract

In an earlier paper, the author used derivatives to construct a deterministic automaton recognizing the language defined by an ω-regular expression. The construction was related to a determinization method invented by Safra. This paper describes a new construction, inspired by Piterman's improvement to Safra's method. It produces an automaton with fewer states.
In addition, the presentation and proofs are simplified by going via a nondeterministic automaton having derivatives as states.

1 Introduction

In 1964, Brzozowski [1] presented an elegant construction leading from a regular expression directly to a deterministic automaton recognizing the language denoted by that expression. The construction, based on the notion of a *derivative*, has been recently re-examined and improved by Owens, Reppy and Turon [2].

The notion of a derivative is easily extended to ω-languages. But, Brzozowski's construction does not work for ω-regular expressions: the automaton constructed from derivatives has, as a rule, too few states.

In an earlier paper [4], the author used derivatives to construct a deterministic automaton recognizing the language defined by an ω-regular expression. The states of this automaton are certain combinations of derivatives. They are analogous to trees appearing in the determinization algorithm invented by Safra [5, 6]. Recently, Piterman [3] improved the Safra's method in a way that reduces the maximum possible number of states from Safra's $(12)^n n^{2n}$ to $2n^n n!$. This paper exploits the Piterman's idea to improve the construction from [4].

It should be noted that, as in [4], we use here a new kind of acceptance condition for ω-automata. All standard acceptance conditions (Büchi, Rabin, Muller, Streett) are defined in terms of visits to *states*. Instead, we use here *transitions* that are (or are not) taken infinitely often. This makes the construction simpler, and further reduces the number of states.

2 Omega-regular languages and their derivatives

We assume a finite *alphabet* Σ of *letters*. A sequence of letters from Σ is called a *word* (over Σ). A word can be finite or infinite, meaning a finite or infinite sequence of letters. The sequence of 0 letters is called the *empty word* and is denoted by ε. The set of all words is denoted by Σ^∞, the set of all finite words by Σ^*, the set of all finite words other than ε by Σ^+, and the set of all infinite words by Σ^ω. Any subset of Σ^∞ is called a *language*.

The *derivative* of a language $X \subseteq \Sigma^\infty$ with respect to a word $w \in \Sigma^*$, denoted by $\partial_w X$, is the set of words obtained by stripping the initial w from words in X starting with w:

$$\partial_w X = \{z \in \Sigma^\infty \mid wz \in X\}\,.$$

(We follow here [2] in using the symbol ∂ for derivative.) Any finite initial portion of a word $x \in \Sigma^\infty$ is called a *prefix* of x. The set of all prefixes of words in a language $X \subseteq \Sigma^\infty$ is denoted by $\mathrm{pref}(X)$. One can easily see that $\partial_w X \neq \varnothing$ if and only if $w \in \mathrm{pref}(X)$.

*Final version appeared in *Fundamenta Informaticae 119*, 3-4 (2012) 393–406.

We use these operations on languages, and assume the reader to be familiar with their properties:

union	$X \cup Y$	for $X \subseteq \Sigma^\infty, Y \subseteq \Sigma^\infty$,
product	$XY = \{xy \mid x \in X,\, y \in Y\}$	for $X \subseteq \Sigma^*, Y \subseteq \Sigma^\infty$,
star	$X^* = \varepsilon \cup X \cup X^2 \cup X^3 \cup \ldots$	for $X \subseteq \Sigma^*$,
omega	$X^\omega = \{x_1 x_2 x_3 \ldots \mid x_i \in X \text{ for } i \geq 1\}$	for $X \subseteq \Sigma^+$.

A *regular language* (over alphabet Σ) is any language constructed from elementary languages $\varnothing$, $\{\varepsilon\}$, and $\{a\}$ for $a \in \Sigma$ by finitely many applications of star, product and union. We assume that regular language is always given by an expression specifying this construction. Such expression is referred to as a *regular expression*. Unless indicated otherwise by means of parentheses, the operators in a regular expression are applied in this order: star, product, union. We customarily omit braces around ε and a when it does not lead to an ambiguity.

An *ω-regular language* is any language of the form

$$\bigcup_{i=1}^{n} X_i\, Y_i^\omega = X_1\, Y_1^\omega \cup X_2\, Y_2^\omega \cup \ldots \cup X_n\, Y_n^\omega, \tag{*}$$

where $n \geq 1$, X_i is a regular language, and Y_i is a nonempty regular language not containing ε, for $1 \leq i \leq n$. Again, we assume that an ω-regular language is always given by an expression of the form (*) or one that can be reduced to that form by means of known properties of the operations. Such an expression is an *ω-regular expression*.

We shall need two helper functions: $\nu(X) = X \cap \{\varepsilon\}$ and $\phi(X) = (\varnothing$ if $X = \varnothing$ or $\{\varepsilon\}$ otherwise). The first is used in the computation of derivatives, and the second to test emptiness of derivatives. They can be computed for any (ω-)regular language by a recursive application of these rules:

$$\begin{array}{lll}
\nu(\varnothing) = \nu(\{a\}) = \varnothing\,, & \nu(X \cup Y) = \nu(X) \cup \nu(Y)\,, & \nu(X^*) = \varepsilon\,, \\
\nu(\varepsilon) = \varepsilon\,, & \nu(XY) = \nu(X)\nu(Y)\,, & \nu(X^\omega) = \varnothing\,. \\[1ex]
\phi(\varepsilon) = \phi(a) = \varepsilon\,, & \phi(X \cup Y) = \phi(X) \cup \phi(Y)\,, & \phi(X^*) = \varepsilon\,, \\
\phi(\varnothing) = \varnothing\,, & \phi(XY) = \phi(X)\phi(Y)\,, & \phi(X^\omega) = \phi(X)\,.
\end{array}$$

Derivatives of an (ω-)regular language are obtained by a recursive application of these rules, where $a, b \in \Sigma$, $a \neq b$, $w \in \Sigma^*$:

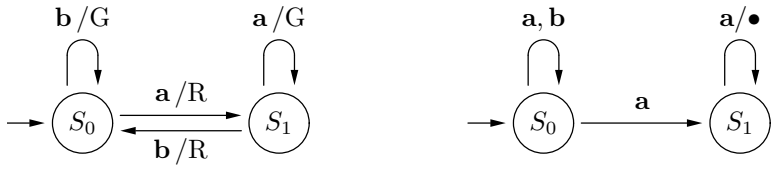
$$\begin{array}{lll}
\partial_\varepsilon X = X\,, & \partial_a\{b\} = \varnothing\,, & \partial_a X^* = (\partial_a X)X^*\,, \\
\partial_a\varnothing = \partial_a\{\varepsilon\} = \varnothing\,, & \partial_a(X \cup Y) = \partial_a X \cup \partial_a Y\,, & \partial_a X^\omega = (\partial_a X)X^\omega\,, \\
\partial_a\{a\} = \varepsilon\,, & \partial_a(XY) = (\partial_a X)Y \cup \nu(X)(\partial_a Y)\,, & \partial_{wa} X = \partial_a(\partial_w X)\,.
\end{array}$$

All these rules except one for X^ω were obtained by Brzozowski in [1]; that for X^ω is obtained in a similar way. One can easily see that the derivative is always an (ω-)regular language. It has been shown in [1] that each regular language has only a finite number n of distinct derivatives, and all of them are found among derivatives with respect to words not longer than $n - 1$.

This result is easily extended to ω-regular languages. Thus, all distinct derivatives of an ω-regular language can be obtained by repeatedly applying the rule $\partial_{(wa)}X = \partial_a(\partial_w X)$ to longer and longer words w until no more distinct derivatives are found. However, the procedure yields ω-regular expressions, and it is often not trivial to decide whether two expressions denote the same language. As shown by Brzozowski, the procedure for a regular language will still end if we can recognize equality using only the basic properties of union (idempotent, commutative, associative). This result is also easily extended to ω-regular languages. It means that we can always effectively obtain expressions for all derivatives, although some of them may denote the same language. We can often eliminate such duplicates using other known properties of the operations.

3 Automata

We define a *finite-state automaton* informally, as a machine that can assume a finite number of distinct *states*. One state is identified as the *initial state*. The machine is started in the initial state and reads an infinite word $w \in \Sigma^\omega$, letter by letter. Each letter causes a *transition* to another, or possibly the same, state. This is usually represented by a graph where nodes represent the states and directed edges represent the transitions, as shown in Fig. 1. Each transition is labeled by the letter that causes it. The initial state is pointed to by an arrow. (The label $\mathbf{a}, \mathbf{b}$ in the Figure is a shorthand for two transitions labeled, respectively, $\mathbf{a}$ and $\mathbf{b}$.)



(a) **Fig. 1** (b)

The automaton is *deterministic* if all transitions from the same state have different labels, as is the case in Fig. 1(a). Otherwise the automaton is *nondeterministic*, as is the case in Fig. 1(b).

The action of the automaton when it reads a word w is represented by a path with labels along the path spelling out the word w. We refer to it as the *run* of the automaton on w. A deterministic automaton has at most one run for a given word; a nondeterministic has usually many such runs.

We say that an automaton *accepts* a word $w \in \Sigma^\omega$ if it has an *accepting run* on w. We say that an automaton *recognizes* a language $X \subseteq \Sigma^\omega$ if it accepts exactly the words belonging to X.

Traditionally, an accepting run is defined in terms of the sequence of states traversed by the automaton. In this paper, we define an accepting run in terms of transitions used by that run. To identify transitions, we attach some arbitrarily chosen tags to them.

In Fig. 1, the tags are shown after a slash. Thus, for the automaton (a) in Fig. 1 we used the tags G and R. We can define an accepting run as one using infinitely many G-transitions and only finitely many R-transitions. One can easily see that the automaton recognizes then the language $(\mathbf{a} \cup \mathbf{b})^*(\mathbf{a}^\omega \cup \mathbf{b}^\omega)$. For the automaton (b) in Fig. 1, we chose $\bullet$ as the only tag. We can define an accepting run to be one that uses the $\bullet$-transition infinitely many times. The automaton recognizes then the language $(\mathbf{a} \cup \mathbf{b})^*\mathbf{a}^\omega$.

It is convenient to say that a transition emits its tag when it occurs. Expressed in this way, an accepting run in case (a) is one that emits G infinitely often and R finitely often, and an accepting run in case (b) is one that emits $\bullet$ infinitely many times.

4 Derivative automaton

Brzozowski's construction [1] is based on this observation: suppose you want to check if the input belongs to given language X. You read a as the first input letter. Then, it remains to check whether the rest belongs to $\partial_a X$. This gives an automaton where each state checks the input for belonging to a specific derivative D of X, and a transition on letter a from that state leads to the state checking for $\partial_a D$.

Unfortunately, this construction does not work for ω-regular languages. Take as an example $X = (\mathbf{a} \cup \mathbf{b})^*(\mathbf{a}^\omega \cup (\mathbf{ab})^\omega)$. We have $\partial_{\mathbf{a}} X = \partial_{\mathbf{b}} X = X$, and the automaton constructed as above has only one state and two transitions. They are too few to specify an accepting run recognizing X. We attack the problem here by distinguishing the derivatives that open the ω-operator.

Let $X = \bigcup_{i=1}^n P_i\, Q_i^\omega$ be any ω-regular language over Σ.

Let $\sharp$ be an arbitrary symbol not in Σ. Define $X' = \bigcup_{i=1}^n P_i(\sharp\, Q_i)^\omega$. This is an ω-regular language over the alphabet $\Sigma \cup \{\sharp\}$.

Let $\mathbb{D}$ be a set of expressions for nonempty derivatives of X' including all derivatives with respect to words over $\Sigma \cup \{\sharp\}$ that do not end with $\sharp$. From the way the derivatives are computed, we have all relations of the form $D_2 = \partial_a D_1$ and $D_2 = \partial_{(\sharp\, a)} D_1$ for $D_1, D_2 \in \mathbb{D}$ and $a \in \Sigma$. Let $\mathbf{D}(X)$ be an automaton defined as follows:

(D1) The states correspond to the elements of $\mathbb{D}$. For $D \in \mathbb{D}$, we use D interchangeably to mean the expression D, the derivative represented by it, and the corresponding state of $\mathbf{D}(X)$.
(D2) For each pair $D_1, D_2 \in \mathbb{D}$ such that $\partial_a D_1 = D_2$ for $a \in \Sigma$, there is a transition $D_1 \xrightarrow{a} D_2$.
(D3) For each pair $D_1, D_2 \in \mathbb{D}$ such that $\partial_{(\sharp\, a)} D_1 = D_2$ for $a \in \Sigma$, there is a transition $D_1 \xrightarrow{a/\bullet} D_2$.
(D4) The initial state is $D_0 = \partial_\varepsilon X' = X'$.
(D5) An accepting run is one that emits $\bullet$ infinitely many times.

The automaton $\mathbf{D}(X)$ is in the following referred to as the *derivative automaton* for X.

As an example, take $X = (\mathbf{a} \cup \mathbf{b})^*(\mathbf{a}^\omega \cup (\mathbf{ab})^\omega)$. We have $X' = (\mathbf{a} \cup \mathbf{b})^*((\sharp\,\mathbf{a})^\omega \cup (\sharp\,\mathbf{ab})^\omega)$. The nonempty derivatives with respect to words not ending with $\sharp$, and the relations between them are:

$$
\begin{aligned}
&D_0 = \partial_\varepsilon X' = X'; && D_0 = \partial_{\mathbf{a}} D_0 = \partial_{\mathbf{b}} D_0;\\
&D_1 = \partial_{\sharp\,\mathbf{a}} X' = (\sharp\,\mathbf{a})^\omega \cup \mathbf{b}\,(\sharp\,\mathbf{ab})^\omega; && D_1 = \partial_{\sharp\,\mathbf{a}} D_0;\\
&D_2 = \partial_{\sharp\,\mathbf{ab}} X' = (\sharp\,\mathbf{ab})^\omega; && D_2 = \partial_{\mathbf{b}} D_1 = \partial_{\mathbf{b}} D_4;\\
&D_3 = \partial_{\sharp\,\mathbf{a}\sharp\,\mathbf{a}} X' = (\sharp\,\mathbf{a})^\omega; && D_3 = \partial_{\sharp\,\mathbf{a}} D_1;\\
&D_4 = \partial_{\sharp\,\mathbf{ab}\sharp\,\mathbf{a}} X' = \mathbf{b}\,(\sharp\,\mathbf{ab})^\omega. && D_4 = \partial_{\sharp\,\mathbf{a}} D_2.
\end{aligned}
$$

The derivative automaton for X is shown in Fig. 2.

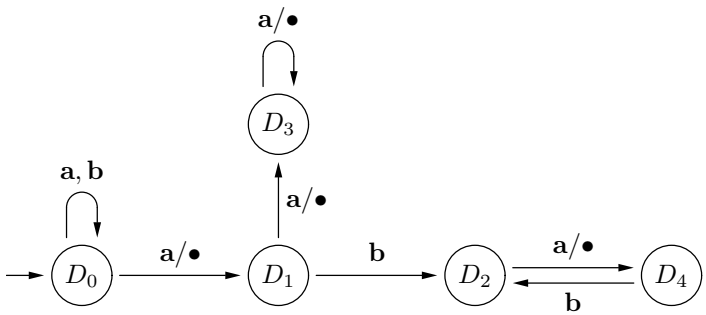


Fig. 2

In a while, we are going to show that $\mathbf{D}(X)$ accepts X. But, one can easily see that the derivative automaton is, as a rule, nondeterministic. In the sections that follow, we shall use it to construct a deterministic automaton for X.

For a path in $\mathbf{D}(X)$, define its *label* to be the sequence of input letters along the path. Define its *extended label* to be the same sequence, with $\sharp$ inserted before each letter that emits $\bullet$.

Lemma 1. *Let v be a word over $\Sigma \cup \{\sharp\}$ not ending with $\sharp$.*
(a) If there exists a path with extended label v from state D_1 to state D_2 of $\mathbf{D}(X)$ then $D_2 = \partial_v D_1$.
(b) If $\partial_v D_1 \neq \varnothing$, there exists a path with extended label v from state D_1 of $\mathbf{D}(X)$ to state $D_2 = \partial_v D_1$.

Proof. By induction on the length of v using (2) and (3) in the definition of $\mathbb{D}$. □

Proposition 1. *The derivative automaton $\mathbf{D}(X)$ accepts word $w \in \Sigma^\omega$ if and only if $w \in X$.*

Proof. (1) Suppose $w \in X$, that is, $w \in P_k Q_k^\omega$ for some k. We have thus $w = pq_1q_2q_3\ldots$ where $p \in P_k$ and $q_i \in Q_k$ for $i \geq 1$. Let $v = p\,\sharp\,q_1\,\sharp\,q_2\,\sharp\,q_3\ldots$. Let $m \geq 1$; denote $v_m = p\,\sharp\,q_1\ldots\sharp\,q_m$. The derivative $\partial_{v_m}(P_k(\sharp\,Q_k)^\omega)$ is nonempty; because $P_k(\sharp\,Q_k)^\omega \subseteq X'$, so is the derivative $\partial_{v_m} X' = \partial_{v_m} D_0$. By Lemma 1b, exists a path from D_0 to $D_m = \partial_{v_m} D_0$ with extended label v_m. Let $v_{m+1} = v_m\,\sharp\,q_{m+1}$. The derivative $\partial_{q_{m+1}} D_m = \partial_{v_{m+1}} X'$ is nonempty for the same reason as $\partial_{v_m} X'$. So, by Lemma 1b exists a path from D_m to $D_{m+1} = \partial_{v_{m+1}} D_m$ with extended label q_{m+1}. It is a continuation of the path to D_m and its label. By repeating this step, we obtain an infinite path from D_0 with extended label v, and non-extended label w. Each letter in that path that follows $\sharp$ emits $\bullet$. It is thus an accepting run on w.

(2) Suppose there is an accepting run on w – a path from D_0 with label w. Let v be the extended label of this path. It contains infinitely many occurrences of $\sharp$, so $v = u_1\,\sharp\,u_2\,\sharp\,u_3\ldots$. For some $m \geq 1$, let $v_m = u_1\,\sharp\,u_2\,\sharp\ldots\sharp\,u_m\,\sharp\,a$, where a is the first letter of u_{m+1}. Word v_m is the extended label of a path

that leads from D_0 to some state D_m. By Lemma 1a, $D_m = \partial_{v_m} D_0 = \partial_{v_m} X'$. As D_m is included among states of $\mathbb{D}$, this derivative is nonempty, so $v_m \in \mathrm{pref}(X')$. It means that there exists at least one k, $1 \geq k \geq n$, such that $v_m \in \mathrm{pref}(P_k(\sharp\, Q_k)^\omega)$. Let K_m be the set of all such k. As this set is nonempty for all $m \geq 1$, one value k must appear in all K_m for $m \geq 1$. Let now k be such value, so for every $m \geq 1$, $v_m = u_1 \sharp\, u_2 \sharp \ldots \sharp\, u_m \sharp\, a$ is a prefix of some word in $P_k(\sharp\, Q_k)^\omega$. This is only possible if $u_1 \in P_k$ and $u_i \in Q_k$ for $i > 1$. Hence, $w = u_1 u_2 u_3 \ldots \in P_k Q_k^\omega \subseteq X$.

□

5 The run tree

All the possible actions of $\mathbf{D}(X)$ when reading a given word can be represented by a directed tree like this in Fig. 3, which shows the possible actions of the automaton from Fig. 2 when reading the infinite word $w = \mathbf{ababaa}\ldots$.

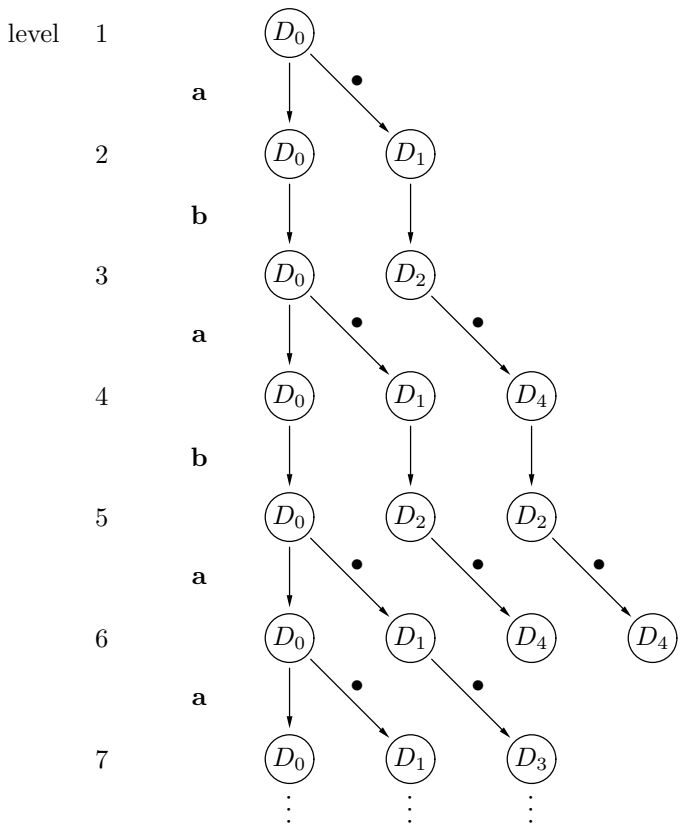


Fig. 3

The Figure shows that, when reading **a** in the initial state D_0, $\mathbf{D}(X)$ can either take a transition back to D_0 without emitting •, or a transition to D_1 and emit •. These transitions are represented by the directed edges in Fig. 3, with • marking the transition that emitted •. The nodes at level 2 represent the possible states after reading the first letter. When reading **b** in any of these states, the automaton has no choice, and takes a transition to D_0, respectively D_2. These states, appearing at level 3, are the possible states of $\mathbf{D}(X)$ after reading the first two letters. In general, the nodes at level l represent the possible states after reading $l-1$ letters, resulting from all possible choices made on the way.

In the following, we refer to a tree like that in Fig. 3 as a *run tree*, and denote by $\mathbf{T}(X, w)$ the run tree of $\mathbf{D}(X)$ for a word $w \in \Sigma^\omega$. One can easily see that the automaton accepts w if an only if $\mathbf{T}(X, w)$ contains a path marked with infinitely many •. In the following, we call it a *live path*.

We need a way to decide if $\mathbf{T}(X, w)$ contains live path. To do this, you cannot just go down from the root and choose an edge marked with • whenever available. It may lead to a dead end such as D_4 on level 6, while a more careful choice would allow you to continue indefinitely. The problem of detecting

a live path in a tree is discussed in some detail in Section 5 of [4]. We adopt the method used there that consists of enclosing the nodes on each level of $\mathbf{T}(X, w)$ in pairs of brackets. The general idea is that a pair of brackets encloses descendants of some common ancestor. (We assume the tree drawn so that the descendants of each node appear contiguously under that node.) Additional nested brackets are introduced around each node that has been reached via an edge marked with $\bullet$. When all inner nodes for some pair of brackets have additional brackets around them, we remove these brackets and signal a "green event" for the outer pair.

If the same pair of brackets has a green event infinitely often, we have a live path. The problem is to identify "the same pair". We use for this purpose numbers on the brackets. As the numbers should not grow indefinitely, they have to be reused after the enclosed node does not have any descendants. This is a "red event" for the pair, indicating that its number will be used in the future for entirely different pair.

As the tree branches out indefinitely, some levels will contain multiple occurrences of the same state, like D_2 on levels 5. The subtrees rooted at these duplicate occurrences are isomorphic, so we need to investigate only one of them. We are thus going to prune the tree, leaving only one of the duplicate states. This leaves a tree $\overline{\mathbf{T}}(X, w)$ with finite width, not exceeding the number of states of $\mathbf{D}(X)$. One can easily see that $\overline{\mathbf{T}}(X, w)$ contains a live path if and only if $\mathbf{T}(X, w)$ does.

As shown in [4], the choice of the branch to keep is very important. A wrong choice may result in missing the live path. In the method described there, we used a left-to-right ordering of the run tree, and always kept the rightmost occurrence. That method was related to the determinization algorithm invented by Safra [5, 6]. (The idea of green events is borrowed from there.)

Safra's algorithm has been recently improved by Piterman [3]. We exploit his idea by using numbers on the brackets to decide which occurrences to remove. We no longer need to keep information about the order; at the end, this reduces the number of states of the deterministic automaton.

6 Annotating the run tree

The result of pruning the tree of Fig. 3 and inserting the brackets according to our method is shown in Fig. 4. Before presenting the exact algorithm, we describe some points in an informal way. of marking and pruning the run tree.

As one can see, it is quite inconvenient to insert brackets in the actual tree. We can do it instead on a copy of each level, as shown on the right. The process can be thus reduced to annotating the tree with strings of state names and numbered brackets. One can, in fact, compute the annotations without ever looking at the tree, and we are going to describe exactly how to do it. Before doing this, let us pay attention to some steps in the process

We notice first that the subtree in brackets numbered 2, with root D_1 at level 2 had a green event at level 4. The $\bullet$-transition from D_2 on level 3 added a pair of brackets around D_4 at level 4. As the only state within the 2-brackets got additional brackets, this was a green event. The additional brackets were removed, and a green event for 2-brackets is indicated by $\Rightarrow$ G 2. We have then an identical situation at level 6.

At level 5, we had initially two occurrences of D_2. Our method uses the *nesting pattern* of the node to decide which one to prune. The nesting pattern is the sequence of numbers on the brackets enclosing the node, written from outside in. The nesting pattern of the leftmost D_2 was thus $[1, 3]$, and that of the rightmost D_2 was $[1, 2]$. Our rule is to retain the node with the lexicographically lowest nesting pattern, so the leftmost D_2 was removed. In this situation, the D_1 on level 4 was left without descendants, and the brackets numbered 3 disappear on level 5. This is a red event for 3-brackets, indicated by $\Rightarrow$ R 3.

When comparing the nesting patterns of different lengths, we always think of the shorter one as extended with very high numbers at the end so that, in particular, $[1, 2] > [1, 2, 3]$.

The exact procedure for annotating the tree is given in the next section, and its application to levels 3 and 4 above is illustrated in detail in Section 6.2.

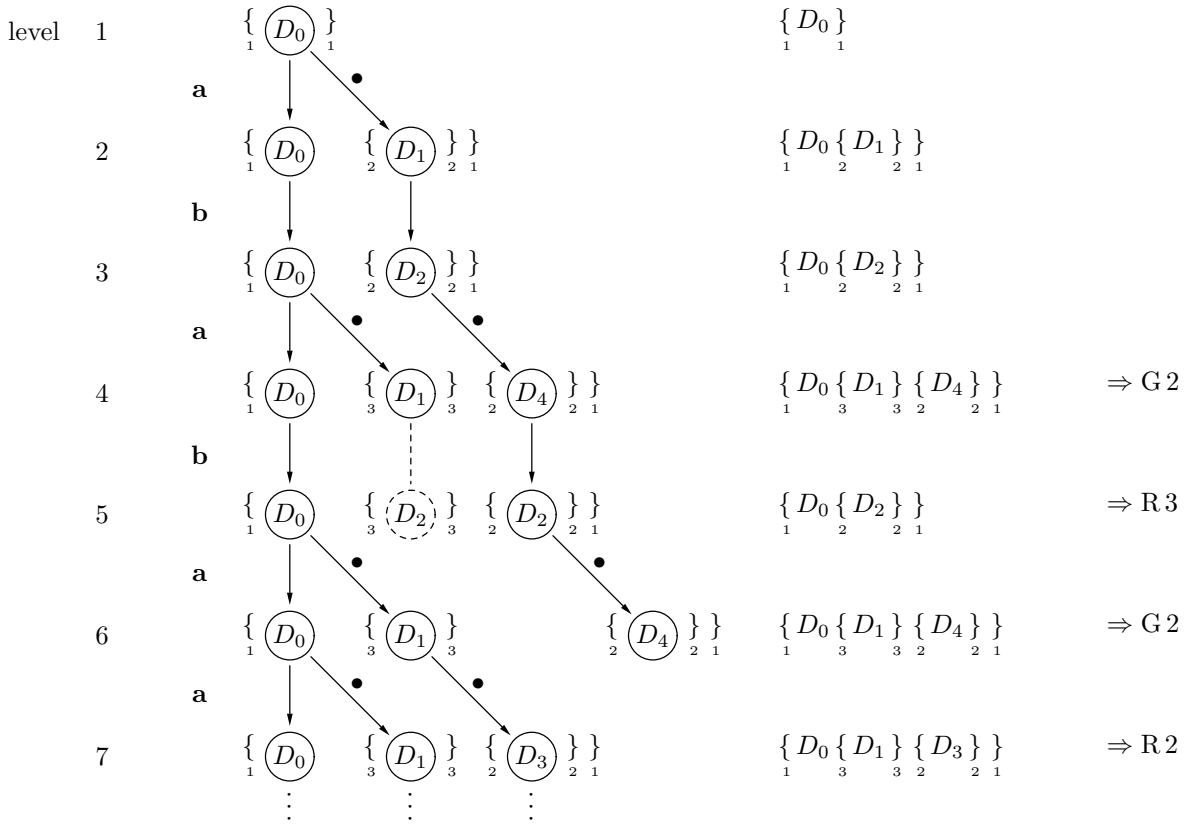


Fig. 4

6.1 Algorithm A

Annotate level 1 with $\underset{1}{\{}\, D_0 \,\underset{1}{\}}$.

For $l > 1$, copy the part of the annotation between, and including, the brackets numbered 1 from level $l-1$ to level l and transform it as follows:

(A1) Replace each D_i by $\partial_a D_i \{\, \partial_{(\sharp\, a)} D_i \,\}$. Omit $\partial_a D_i$ if it is empty, and $\{\, \partial_{(\sharp\, a)} D_i \,\}$ if $\partial_{(\sharp\, a)} D_i = \varnothing$. Assign the lowest unused numbers to the newly introduced brackets.

(A2) If a node occurs more than once, remove all its occurrences except the one with lexicographically lowest nesting pattern. If more nodes have identical lowest pattern, choose any one of them.

(A3) Remove all pairs of brackets that do not contain any nodes.
Set r to the lowest number on the removed pair, or to $N+1$ if there were none, where N is the number of states of $\mathbf{D}(X)$.

(A4) Call a pair of brackets "green" if all states inside it are enclosed in additional nested brackets. Remove all brackets (but not nodes) inside each green pair.
Set g to the lowest number on the green pair, or to $N+1$ if there were none.

(A5) For pair numbered m, define $\mathrm{rem}(m)$ to be the number of pairs removed in step (A3) that had number lower than m. Change the number of each pair from m to $m - \mathrm{rem}(m)$.

(A6) If $g < r$, append $\Rightarrow \mathrm{G}\, g$ on the right. If $r \leq g$ and $r \neq N+1$, append $\Rightarrow \mathrm{R}\, r$.

6.2 Examples

Algorithm A is illustrated below by applying it to the annotations on levels 3 and 4 of Fig. 3.

	input **a** applied to level 3	input **b** applied to level 4
start with	$\underset{1}{\{}\, D_0 \underset{2}{\{}\, D_2 \underset{2}{\}}\, \underset{1}{\}}$	$\underset{1}{\{}\, D_0 \underset{3}{\{}\, D_1 \underset{3}{\}}\, \underset{2}{\{}\, D_4 \underset{2}{\}}\, \underset{1}{\}}$
after (A1)	$\underset{1}{\{}\, D_0 \underset{4}{\{}\, D_1 \underset{4}{\}}\, \underset{2}{\{}\, \underset{3}{\{}\, D_4 \underset{3}{\}}\, \underset{2}{\}}\, \underset{1}{\}}$	$\underset{1}{\{}\, D_0 \underset{3}{\{}\, D_2 \underset{3}{\}}\, \underset{2}{\{}\, D_2 \underset{2}{\}}\, \underset{1}{\}}$
after (A2)	$\underset{1}{\{}\, D_0 \underset{4}{\{}\, D_1 \underset{4}{\}}\, \underset{2}{\{}\, \underset{3}{\{}\, D_4 \underset{3}{\}}\, \underset{2}{\}}\, \underset{1}{\}}$	$\underset{1}{\{}\, D_0 \underset{3}{\{}\, \underset{3}{\}}\, \underset{2}{\{}\, D_2 \underset{2}{\}}\, \underset{1}{\}}$
after (A3)	$\underset{1}{\{}\, D_0 \underset{4}{\{}\, D_1 \underset{4}{\}}\, \underset{2}{\{}\, \underset{3}{\{}\, D_4 \underset{3}{\}}\, \underset{2}{\}}\, \underset{1}{\}}$ $\quad r = 6$	$\underset{1}{\{}\, D_0 \underset{2}{\{}\, D_2 \underset{2}{\}}\, \underset{1}{\}}$ $\quad r = 3$
after (A4)	$\underset{1}{\{}\, D_0 \underset{4}{\{}\, D_1 \underset{4}{\}}\, \underset{2}{\{}\, D_4 \underset{2}{\}}\, \underset{1}{\}}$ $\quad g = 2$	$\underset{1}{\{}\, D_0 \underset{2}{\{}\, D_2 \underset{2}{\}}\, \underset{1}{\}}$ $\quad g = 6$
after (A5)	$\underset{1}{\{}\, D_0 \underset{3}{\{}\, D_1 \underset{3}{\}}\, \underset{2}{\{}\, D_4 \underset{2}{\}}\, \underset{1}{\}}$	$\underset{1}{\{}\, D_0 \underset{2}{\{}\, D_2 \underset{2}{\}}\, \underset{1}{\}}$
after (A6)	$\underset{1}{\{}\, D_0 \underset{3}{\{}\, D_1 \underset{3}{\}}\, \underset{2}{\{}\, D_4 \underset{2}{\}}\, \underset{1}{\}}$ $\quad \Rightarrow \mathrm{G}\,2$	$\underset{1}{\{}\, D_0 \underset{2}{\{}\, D_2 \underset{2}{\}}\, \underset{1}{\}}$ $\quad \Rightarrow \mathrm{R}\,3$

6.3 Analysis

Before we prove the exact condition for the existence of a live path, we shall need some auxiliary facts.

One can easily see that (A1)–(A2) produce a correct image of nodes on level l with brackets from level $l-1$ carried down to enclose descendants of the contained nodes, and with additional brackets around nodes reached by marked edges.

We shall say that a node, or a pair of brackets, *resides in* a pair B of brackets to mean that B is the nearest enclosing pair.

Lemma 2. *The annotation for each level $l \geq 1$ has the following properties:*

(L1) Each pair of brackets has at least one node residing in it.

(L2) There are at most N pairs of brackets, where N is the number of nodes of $\mathbf{D}(X)$.

(L3) The brackets are numbered with consecutive numbers $1, \ldots m$.

(L4) Inner brackets have numbers higher than enclosing brackets.

Proof. The Proposition is obviously true for level 1. Suppose it is true for level $l \geq 1$. We show that (L1)–(L4) remain true for level $l+1$.

(L1) Empty pairs of brackets created by (A1) and (A2) are removed by (A3). No nodes reside in a pair that became green after (A1). But, (A4) removes all its inner brackets so that all contained nodes reside in that pair.

(L2) is an immediate consequence of (L1) and the fact that (A2) removes all duplicate nodes.

(L3) This is the obvious result of (A5).

(L4) New brackets are added in (A1). Each added pair obtains the lowest unused number. Because of this and (L3), this number is higher than that on the enclosing brackets. Renumbering in (A5) cannot change this: suppose brackets numbered m_1 enclose brackets numbered $m_2 > m_1$. If (A3) removed k empty pairs of brackets with numbers lower than m_1, both m_1 and m_2 are reduced by k. If (A3) removed k pairs with numbers between m_1 and m_2, m_2 is reduced by k. But, k must be less than $m_2 - m_1 - 1$, so $m_2 > m_1$ remains true. Removing pairs with numbers higher than m_2 leaves m_1 and m_2 unchanged. □

6.4 Do we have a live path?

Proposition 2. *$\overline{\mathbf{T}}(X, w)$ contains a live path if and only if there exists g such that $\Rightarrow G\,g$ appears on infinitely many levels, while $\Rightarrow R\,r$ with $r \leq g$ appears on at most finitely many levels.*

Proof. (1) Suppose $\Rightarrow \mathrm{G}\,g$ appears on infinitely many levels. Suppose further that no $\Rightarrow \mathrm{R}\,r$ with $r \leq g$ appears after some level l_0. It means that after that level, we have a pair numbered g that is neither removed nor renumbered, and nodes within that pair form a subtree of $\overline{\mathbf{T}}(X, w)$ rooted at some node D_x.

The $\Rightarrow$ G g appearing on infinitely many levels means that the subtree is infinite. By König's lemma, it contains an infinite path p from D_x.

Consider any level $l > l_0$ where the pair g becomes green. Step (A4) removed all brackets nested inside it. On each subsequent level, step (A1) adds a pair of brackets around each node reached by an edge marked with $\bullet$. The next green level is achieved when each node within the g-brackets have been reached by a path containing at least one marked edge. Thus, the part of p between any two green levels contains at least one marked edge. As there are infinitely many green levels, p contains infinitely marked edges. The path from the root of $\overline{\mathbf{T}}(X, w)$ to D_x, extended with p is a live path in $\overline{\mathbf{T}}(X, w)$.

(2) Suppose $\overline{\mathbf{T}}(X, w)$ contains a live path p. Each level l of the tree contains a node $D_p(l)$ belonging to p. Each pair of brackets enclosing $D_p(l)$ encloses a subtree with some node on p as its root. As numbers on the brackets change in the process, we shall use this node to identify the brackets. The nested brackets enclosing $D_p(l)$ are thus described by the sequence $B(l) = B_0, B_1, \ldots, B_n$ of nodes on p that are roots of the nested subtrees. The nodes are listed top down, so that B_0 is always the root of $\overline{\mathbf{T}}(X, w)$ and represents the pair numbered 1. The sequence of numbers on these brackets is the nesting pattern of $D_p(l)$.

Suppose that pair B_0 becomes green at infinitely many levels, so $\Rightarrow$ G 1 appears infinitely often. The pair is never removed, so $\Rightarrow$ R 1 never appears, and the condition of the theorem is satisfied.

Otherwise, B_0 becomes green at only finitely many levels. Let us consider a more general situation where $k \geq 1$ outermost pairs: $B_0, B_1, \ldots, B_{k-1}$ no longer turn green after some level l_0. At some level $l_1 > l_0$, the first marked edge on p adds a pair B_k inside B_{k-1}. As the result of pruning, the number k may be reduced (but never to 0), and the pair B_k replaced by another one. In order not to complicate the notation, we shall consider a generic situation with $k+1$ outermost brackets $B_0, B_1, \ldots, B_{k-1}, B_k$ of which the first k no longer turn green.

Let us see in detail what happens when Algorithm A is applied to a level $l \geq l_1$.

(A1) If the next edge of p is marked, (A1) adds a pair of brackets around $D_p(l)$, extending $B(l)$ with one element. It does not affect the nest $B_0, B_1, \ldots, B_k$.

(A2) Pruning replaces $D_p(l)$ by another node $D'_p(l)$. We have a new live path p', that goes from the root of $\overline{\mathbf{T}}(X, w)$ to $D'_p(l)$, and then continues with a copy of p starting at $D'_p(l)$. The node $D'_p(l)$ will now, in general, be enclosed in a new nest of brackets, represented by a new sequence $B'(l)$ of root nodes on p'. The paths p and p' have a common initial portion (at least the starting node), so $B'(l)$ contains $k' \leq k$ initial elements of $B(l)$ that no more turn green. We end thus up with $B'(l) = B_0, \ldots, B_{k'-1}, B'_k, \ldots$ where $B_0, B_1, \ldots, B'_{k'-1}$ no more turn green. By the choice of $D'_p(l)$, the nesting pattern of $B'(l)$ is not higher than that of $B(l)$, and so is the nesting pattern of B'_k.

(A3) All elements of $B(l)$ contain a node from p, so they are not deleted by (A3).

(A4) At a green event, (A4) removes a tail of $B(l)$ after the pair that turned green. As $B_0, B_1, \ldots, B_{k-1}$ do not turn green, it leaves $B_0, B_1, \ldots, B_{k-1}, B_k$ unchanged.

(A5) The renumbering by (A5) does not change $B(l)$, but reduces some items of $P(l)$, resulting in a lower nesting pattern.

In each case, we are left with a nest $B(l) = B_0, B_1, \ldots, B_k, \ldots$ of brackets of which the first k no more turn green. The number k may be reduced, but the nesting pattern of B_k, if it changes, can only be lowered by (A2) or (A5). As the nesting pattern cannot be indefinitely lowered, it will not be changed any more after some level l_2. Consider any level $l > l_2$ and suppose B_k turns green infinitely often after this level. As it is not deleted by (A3) or (A4), and no longer replaced by (A2) or renumbered by (A5), $\Rightarrow$ G g with the number g of B_k appears infinitely often. The absence of renumbering by (A5) means that no pair with number $r \leq g$ is deleted by (A3) so $\Rightarrow$ R r with $r \leq g$ no longer appears. The condition of the theorem is thus satisfied.

If B_k does not turn green infinitely often, it does so for the last time at some level l_3, resulting in $B(l_3) = B_0, B_1, \ldots, B_k$. The first marked edge on p changes this to $B_0, B_1, \ldots, B_k, B_{k+1}$. We have now a nest of $k+2$ brackets of which the first $k+1$ do not turn green any more. The reasoning can now be repeated. However, the nesting pattern of B_{k+1} is lower than that of B_k, that was possibly lowered, but never increased, in the previous turn. According to (L1)–(L4), the nesting pattern cannot be lowered beyond $[1, 2, \ldots, N]$, after which every marked edge on p produces a green event. $\square$

7 Towards a deterministic automaton

According to Proposition 2, we can decide whether $\mathbf{T}(X, w)$ contains a live path by annotating it according to Algorithm A.

In fact, we do not need to construct $\mathbf{D}(X)$ and $\mathbf{T}(X, w)$; we can use Algorithm A to directly compute the annotations. Moreover, we can construct an automaton that will emit the outputs Gg and Rr appearing in the annotations while reading the word w.

Define the "state" to be the part of the annotation between, and including, the brackets numbered 1. Define the "output" to be the part to the right of $\Rightarrow$, if any. At each level, the next state and the output are determined by the state at that level and the next letter of w.

The number of different states is finite: as a consequence of (A2) the state cannot contain more than N nodes and, according to (L2), it cannot contain more than N bracket pairs, where N is the number of nodes of $\mathbf{D}(X)$.

The number of states being finite, we can use (A1)–(A6) to compute once for all the next state and output for each state and input letter.

The exact construction of the automaton for an expression X is given below.

7.1 The construction

Let $X = \bigcup_{i=1}^{n} P_i Q_i^{\omega}$ be a given ω-regular expression over alphabet Σ. Define $X' = \bigcup_{i=1}^{n} P_i(\sharp Q_i)^{\omega}$, where $\sharp$ is a new letter not in Σ. Compute all recognizably distinct expressions for derivatives of X' with respect to words not ending with $\sharp$. Using the property ϕ defined in Section 2, identify and discard those denoting $\varnothing$. Denote the set of the remaining expressions by $\mathbb{D}$. Denote the elements of $\mathbb{D}$ by $D_0, \ldots, D_n$. Find all relations $D_i = \partial_a D_j$ and $D_i = \partial_{(a\,\sharp)} D_j$ for $D_i, D_j \in \mathbb{D}$, $a \in \Sigma$.

Define the automaton $\mathbf{A}(X)$ as follows.

(1) The states of the automaton are sequences consisting of symbols $D_0, \ldots, D_n$ and numbered brackets. To obtain all states, start with the initial state defined by (2), and apply (3) to construct states reached after 1, 2, 3, etc., letters until no new states are obtained.
Note that the order of D_i's does not matter when comparing states. Two states are identical if they contain the same D_i's with the same nesting patterns.

(2) The initial state is $\underset{1}{\{}\, \partial_\varepsilon X' \,\underset{1}{\}}$.

(3) Transitions: for a state s and an input letter $a \in \Sigma$, apply (A1)–(A6) to s. The part of the result between, and including, the brackets numbered 1 is the next state. The output is to the right of $\Rightarrow$ (if any).

(4) Acceptance condition: a word $w \in \Sigma^{\omega}$ is accepted by $\mathbf{A}(X)$ if and only if there exists g such that the automaton applied to w emits Gg infinitely many times, and emits any Rr with $r \leq g$ only finitely many times.

Proposition 3. *The automaton $\mathbf{A}(X)$ recognizes the language X.*

Proof. One can easily see that the states and the outputs of $\mathbf{A}(X)$ when it reads a word w are exactly the annotations produced by Algorithm A for $\mathbf{T}(X, w)$. According to Proposition 2, $\mathbf{A}(X)$ accepts w if and only if $\mathbf{T}(X, w)$ contains a live path, and $\mathbf{T}(X, w)$ contains a live path if an only if $\mathbf{D}(X)$ accepts w. According to Proposition 1, $\mathbf{D}(X)$ accepts w if and only if $w \in X$.

□

7.2 Example

The states and transitions for $X = (\mathbf{a} \cup \mathbf{b})^*(\mathbf{a}^\omega \cup (\mathbf{ab})^\omega)$ are shown below.

state	next state for **a**	next state for **b**
$A = \underset{1}{\{}\, D_0 \,\underset{1}{\}}$	$B = \underset{1}{\{}\, D_0 \underset{2}{\{}\, D_1 \,\underset{2}{\}}\, \underset{1}{\}}$	$A = \underset{1}{\{}\, D_0 \,\underset{1}{\}}$
$B = \underset{1}{\{}\, D_0 \underset{2}{\{}\, D_1 \,\underset{2}{\}}\, \underset{1}{\}}$	$C = \underset{1}{\{}\, D_0 \underset{3}{\{}\, D_1 \,\underset{3}{\}}\, \underset{2}{\{}\, D_3 \,\underset{2}{\}}\, \underset{1}{\}} \Rightarrow$ G2	$D = \underset{1}{\{}\, D_0 \underset{2}{\{}\, D_2 \,\underset{2}{\}}\, \underset{1}{\}}$
$C = \underset{1}{\{}\, D_0 \underset{3}{\{}\, D_1 \,\underset{3}{\}}\, \underset{2}{\{}\, D_3 \,\underset{2}{\}}\, \underset{1}{\}}$	$C = \underset{1}{\{}\, D_0 \underset{3}{\{}\, D_1 \,\underset{3}{\}}\, \underset{2}{\{}\, D_3 \,\underset{2}{\}}\, \underset{1}{\}} \Rightarrow$ G2	$D = \underset{1}{\{}\, D_0 \underset{2}{\{}\, D_2 \,\underset{2}{\}}\, \underset{1}{\}} \Rightarrow$ R2
$D = \underset{1}{\{}\, D_0 \underset{2}{\{}\, D_2 \,\underset{2}{\}}\, \underset{1}{\}}$	$E = \underset{1}{\{}\, D_0 \underset{3}{\{}\, D_1 \,\underset{3}{\}}\, \underset{2}{\{}\, D_4 \,\underset{2}{\}}\, \underset{1}{\}} \Rightarrow$ G2	$A = \underset{1}{\{}\, D_0 \,\underset{1}{\}} \Rightarrow$ R2
$E = \underset{1}{\{}\, D_0 \underset{3}{\{}\, D_1 \,\underset{3}{\}}\, \underset{2}{\{}\, D_4 \,\underset{2}{\}}\, \underset{1}{\}}$	$C = \underset{1}{\{}\, D_0 \underset{3}{\{}\, D_1 \,\underset{3}{\}}\, \underset{2}{\{}\, D_3 \,\underset{2}{\}}\, \underset{1}{\}} \Rightarrow$ R2	$D = \underset{1}{\{}\, D_0 \underset{2}{\{}\, D_2 \,\underset{2}{\}}\, \underset{1}{\}} \Rightarrow$ R3

The resulting automaton $\mathbf{A}(X)$ is shown in Fig. 4. It accepts the input word if it emits G2 infinitely often and R2 only finitely often.

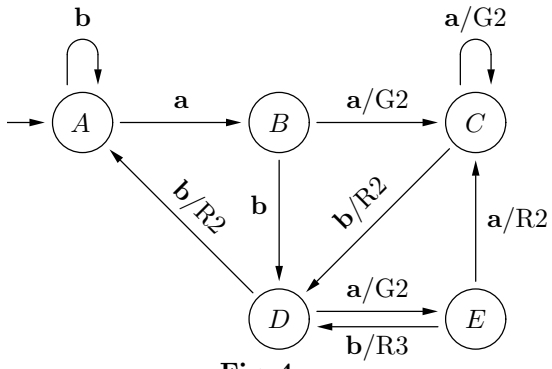


Fig. 4

8 Comments

By eliminating the ordering of nodes, the described construction gives fewer states than one in [4], but their number is still exponential. The construction and proofs have been simplified by introducing the derivative automaton and the run tree.

The annotations used as states of $\mathbf{A}(X)$ are essentially isomorphic with the Piterman's trees obtained for the derivative automaton $\mathbf{D}(X)$. The author believes that annotating the run tree with bracket structures instead of constructing Piterman's trees makes the process simpler to define and visualize: the bracket structures are easier to represent in print than trees, In addition, they can be computed directly without constructing the derivative automaton.

The acceptance condition in terms of "green" and "red" outputs can be converted to parity condition similar to that in [3]. Replace the output Gg by $2g$ and Rr by $2r - 1$; a run is then accepting if and only if the lowest number emitted infinitely often is even.

Estimating the number of states in the same way as in [3] gives $n^n(n-1)!$ as the upper bound. It is lower than that in [3] by the factor of $2n$ due to acceptance condition in terms of transitions rather than states. According to the above result, the maximum possible number of states of $\mathbf{A}(X)$ for $\mathbf{D}(X)$ with 5 states is $5^5(5-1)! = 75000$. Is the fact that we obtained $\mathbf{A}(X)$ with only 5 states just a lucky coincidence, or is there something about the derivative automaton that can be used to lower the upper bound?

To close, we repeat the final remark from [4], pointing out two properties of Brzozowski's construction that our construction fails to achieve.

First, Brzozowski can produce a unique, minimal, automaton for the given language, not depending on the expression used to denote that language. This happens when all derivatives used in the construction denote distinct languages. Our automaton depends strongly on the expression, and need not have the minimal number of states. (In fact, a unique minimal automaton does not in general exist for an ω-regular language.)

Second, Brzozowski's construction can be applied to extended regular expressions, that allow intersection and complementation in addition to union, product, and star. (It was, in fact, presented so in [1].) Unfortunately, our Proposition 1 breaks down if X does not have the stated form.

References

[1] Brzozowski, J. A. Derivatives of regular expressions. *Journal of the ACM 11*, 4 (Oct. 1964), 481–494.

[2] Owens, S., Reppy, J., and Turon, A. Regular-expression derivatives re-examined. *Journal of Functional Programming 19*, 2 (2009), 173–190.

[3] Piterman, N. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Logical Methods in Computer Science 3*, 3 (2007).

[4] Redziejowski, R. R. Construction of a deterministic ω-automaton using derivatives. *Informatique Théorique et Applications 33* (1999), 133–158.

[5] Safra, S. On the complexity of ω-automata. In *Proc. 29th Annual Symposium on Foundations of Computer Science* (1988), IEEE, pp. 319–327.

[6] Safra, S. Complexity of automata on infinite objects. Master's thesis, Weizmann Institute of Science, Rehovot, Israel, Mar. 1989.