An Improved Construction of Deterministic Omega-automaton using Derivatives

Roman R. Redziejowski

Giraf's Research
roman.redz@swipnet.se

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 [5], the author used derivatives to construct a deterministic automaton recognizing the language defined by an ω-regular expression. While each state of Brzozowski's automaton corresponds to one derivative, the states in [5] are certain combinations of derivatives. They are analogous to trees appearing in the determinization algorithm invented by Safra [6, 7]. 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 [5].

It should be noted that, as in [5], 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 pref(X). One can easily see that $\partial_w X \neq \varnothing$ if and only if $w \in \mathrm{pref}(X)$.

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	$X^\omega = \{x_1x_2x_3 \ldots \mid x_i \in X \text{ for } i \geq 1\}$	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:

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

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

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 was extended to ω-regular languages in [4]. 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 **a**, **b** in the Figure is a shorthand for two transitions labeled, respectively, **a** and **b**.)

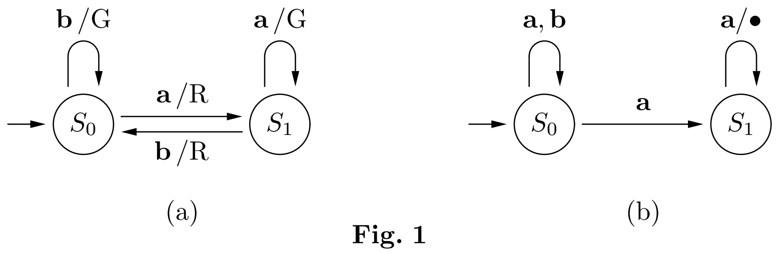

Fig. 1

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 given word $w \in \Sigma^\omega$ if it has an *accepting run* on w.

In this paper, we define an accepting run in terms of transitions used by that run. To identify transitions, we place additional labels on them; in Fig. 1 they are shown following a slash. We can imagine that whenever the automaton executes a transition, it emits the label identifying that transition.

For example, the automaton (a) in Fig. 1 emits G ("green light") when going from S_1 back to S_1 in response to input letter **a**, and R ("red light") when going from S_1 to S_0 in response to letter **b**. We may define in this case an accepting run to be one that emits G infinitely may times, and R only finitely many times.

The automaton (b) in Fig. 1 emits $\bullet$ when going from S_1 to S_1 in response to a. In this case, we may define an accepting run to be one that emits $\bullet$ infinitely many times.

We say that an automaton *recognizes* a language $X \subseteq \Sigma^\omega$ if it accepts exactly the words belonging to X. One can easily see that the automaton (a) in Fig. 1 recognizes $(\mathbf{a} \cup \mathbf{b})^*(\mathbf{a}^\omega \cup \mathbf{b}^\omega)$ and the automaton (b) recognizes $(\mathbf{a} \cup \mathbf{b})^*\mathbf{a}^\omega$.

4 Derivative automaton

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, consider $X = (\mathbf{a} \cup \mathbf{b})^*(\mathbf{a}^\omega \cup (\mathbf{ab})^\omega)$. 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:

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

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

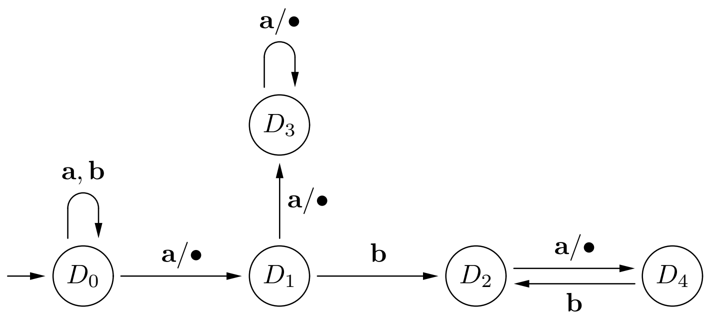

Fig. 2

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 $w = pq_1q_2q_3\dots$ where $p \in P_k$ and $q_i \in Q_k$ for $i \geq 1$. Let $v = p\,\sharp\, q_1 \,\sharp\, q_2 \,\sharp\, q_3 \dots$. Let $m \geq 1$; denote $v_m = p\,\sharp\, q_1 \dots \sharp\, q_m$. 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 for w.

(2) Suppose there is an accepting run for 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 $v = u_1 \,\sharp\, u_2 \,\sharp\, u_3 \dots$. For some $m \geq 1$, let $v_m = u_1 \,\sharp\, u_2 \,\sharp \dots \sharp\, u_m \,\sharp\, a$, 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 $v_m \in \text{pref}(X')$. It means that there exists at least one k, $1 \geq k \geq n$, such that $v_m \in \text{pref}(P_k(\sharp\, Q_k)^\omega)$. 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 $m \geq 1$, $v_m = u_1 \,\sharp\, u_2 \,\sharp \dots \sharp\, u_m \,\sharp\, a$ 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, $w = u_1u_2u_3 \dots \in P_k Q_k^\omega \subseteq X$. □

5 The run DAG

The possible runs of $\mathbf{D}(X)$ on a given word can be represented by a directed acyclic graph (DAG) like this in Fig. 3. Its edges represent transitions, with $\bullet$ marking the transitions that emit $\bullet$.

The graph in Fig. 3 shows all possible runs of the automaton from Fig. 2 on the infinite word $w = \mathbf{aaabab}\dots$. For example, if $\mathbf{D}(X)$ reads letter $\mathbf{a}$ in state D_0 it may either go back to D_0 without emitting $\bullet$, or go to D_1 and emit $\bullet$.

Denote by $\mathbf{G}(X, w)$ the run DAG of $\mathbf{D}(X)$ for a word $w \in \Sigma^\omega$. One can easily see that the automaton accepts w if an only if $\mathbf{G}(X, w)$ contains a path marked with infinitely many $\bullet$. In the following, we call it a *live path*.

We need a way to decide if $\mathbf{G}(X, w)$ contains a live path. To do this, you cannot just go down from the root and choose an edge marked with $\bullet$ whenever available. It may lead to a dead end such as D_3 on level 4, while a more careful choice would allow you to continue indefinitely. The problem of detecting a live path in a graph is discussed in some detail in Section 5 of [5]. We adopt the method used there that consists of enclosing the nodes on each level of $\mathbf{G}(X, w)$ in pairs of numbered brackets. As it is inconvenient to insert those pairs in the actual graph, we do it on a copy of each level, as shown on the right in Fig. 3. The procedure consists thus of annotating the levels of DAG as shown in the Figure.

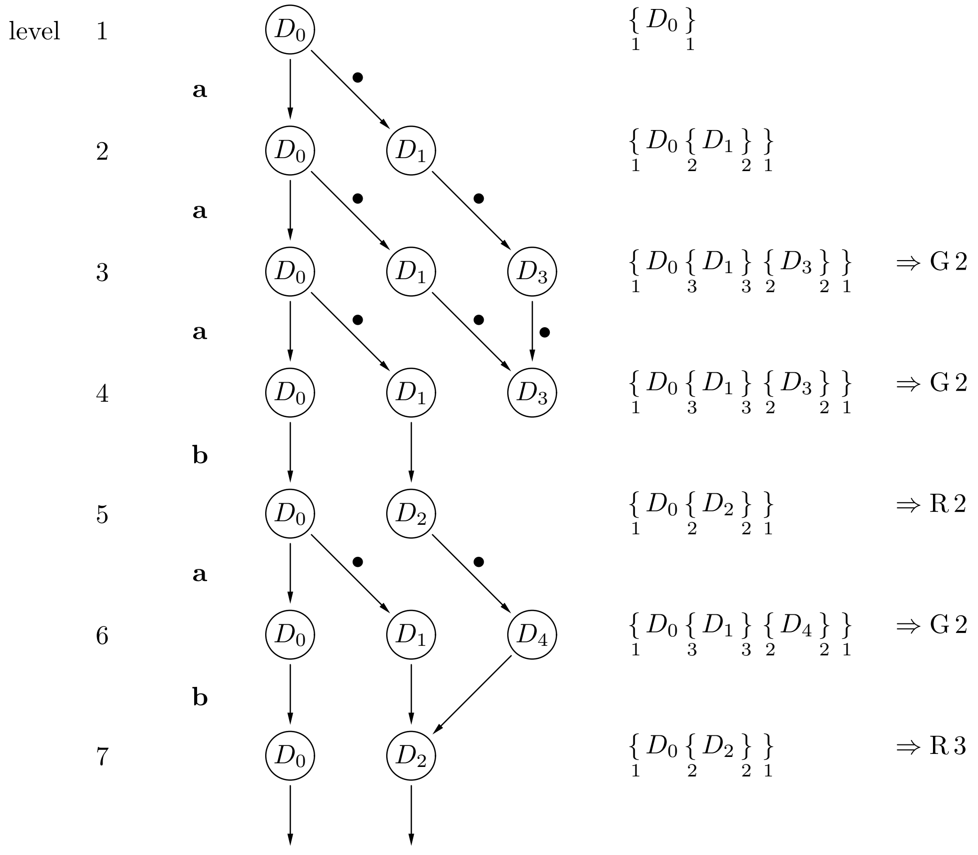
level
1
2
3
4
5
6
7
a
a
a
b
a
b
D_0
D_1
D_2
D_3
D_4
●
$\{_1 D_0 \}_1$
$\{_1 D_0 \{_2 D_1 \}_2 \}_1$
$\{_1 D_0 \{_3 D_1 \}_3 \{_2 D_3 \}_2 \}_1$ ⇒ G 2
$\{_1 D_0 \{_3 D_1 \}_3 \{_2 D_3 \}_2 \}_1$ ⇒ G 2
$\{_1 D_0 \{_2 D_2 \}_2 \}_1$ ⇒ R 2
$\{_1 D_0 \{_3 D_1 \}_3 \{_2 D_4 \}_2 \}_1$ ⇒ G 2
$\{_1 D_0 \{_2 D_2 \}_2 \}_1$ ⇒ R 3

Fig. 3

The general idea is that a pair of brackets encloses descendants of some common ancestor. Additional nested brackets enclose the nodes that have been reached via a path marked with one or more •. When all inner nodes have brackets around them, we remove these brackets and signal a "green event" for the outer pair. Such green events are indicated by ⇒ G 2 in the Figure.

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 the numbers on the brackets. As the numbers should not grow indefinitely, they have to be reused after the enclosed node does not have descendants, such as D_3 on level 4. This is a "red event" for the pair, indicating that its number will be used in the future for entirely different pair. Such a red event is indicated here by ⇒ R 2.

The idea of green events has been borrowed from the determinization algorithm invented by Safra [6,7]. The algorithm keeps track of possible paths in an automaton with the help of tree structures where nodes are sets of states. These trees are analogous to our level annotations.

Safra's algorithm has been recently improved by Piterman [3]. The improvement concerns the situation where several paths reach the same state. Each path has its own history; we have to choose one of them in such a way that we do not miss a possible live path. In order to make this choice, Safra keeps nodes in his trees ordered according to "age"; in [5], the nodes are kept in a left-to-right order. Piterman eliminates the order by numbering the nodes. We follow the idea by exploiting numbers on the brackets. This requires some renumbering as we proceed down the DAG, but, by ignoring the order, it significantly reduces the number of distinct annotations.

6 Annotating the run DAG

To describe the exact procedure for annotating the run DAG and prove facts about it, we shall need some additional terminology.

For a node D_i in a bracket structure, we define its *nesting pattern* as the sequence of numbers on the brackets containing it, from outside in. As an example, in the structure

$$\underset{1}{\{}\, D_0 \,\underset{4}{\{}\, D_1 \,\underset{4}{\}}\, \underset{3}{\{}\, \underset{5}{\{}\, D_2 \,\underset{5}{\}}\, \underset{3}{\}}\, \underset{2}{\{}\, \underset{6}{\{}\, D_3 \,\underset{6}{\}}\, \underset{2}{\}}\, \underset{1}{\}}$$

the nesting pattern of D_1 is (1 - 4), that of D_2 is (1 - 3 - 5), and that of D_3 is (1 - 2 - 6).

We compare nesting patterns according to the first different position. Thus, (1 - 2 - 6) is lower than (1 - 3 - 5). An empty position is considered higher than any number, so (1 - 2 - 6) is lower than (1 - 2).

We 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. Thus, in the above example, both D_0 and the pair numbered 3 reside in the pair numbered 1.

6.1 The 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:

$$\begin{array}{lll}
\partial_a D_i \,\{\, \partial_{(\sharp\, a)} D_i \,\} & \text{if} & \partial_a D_i \neq \varnothing \;\text{ and }\; \partial_{(\sharp\, a)} D_i \neq \varnothing, \\
\{\, \partial_{(\sharp\, a)} D_i \,\} & \text{if} & \partial_a D_i = \varnothing \;\text{ and }\; \partial_{(\sharp\, a)} D_i \neq \varnothing, \\
\partial_a D_i & \text{if} & \partial_a D_i \neq \varnothing \;\text{ and }\; \partial_{(\sharp\, a)} D_i = \varnothing, \\
\text{empty string} & \text{if} & \partial_a D_i = \varnothing \;\text{ and }\; \partial_{(\sharp\, a)} D_i = \varnothing,
\end{array}$$

where a is the input letter. Each time assign the lowest unused number to the brackets.

(A2) If a node appears more than once, remove all its occurrences except the one with lowest nesting pattern. If more than one have identical lowest pattern, choose 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 $\text{rem}(m)$ 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 $m - \text{rem}(m)$.

(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 Example

We illustrate the steps (A1)–(A6) by applying them to the annotation appearing on levels 3 and 4 of Fig. 3. Note that after (A1) for input **a**, we have two occurrences of D_3, with nesting patterns (1 - 3 - 5) and (1 - 2 - 6), respectively. The latter is retained as having the lower nesting pattern.

step	level 3: input **a**	level 4: input **b**
	$\underset{1}{\{} D_0 \underset{3}{\{} D_1 \underset{3}{\}} \underset{2}{\{} D_3 \underset{2}{\}} \underset{1}{\}}$	$\underset{1}{\{} D_0 \underset{3}{\{} D_1 \underset{3}{\}} \underset{2}{\{} D_3 \underset{2}{\}} \underset{1}{\}}$
$(A1)$	$\underset{1}{\{} D_0 \underset{4}{\{} D_1 \underset{4}{\}} \underset{3}{\{} \underset{5}{\{} D_3 \underset{5}{\}} \underset{3}{\}} \underset{2}{\{} \underset{6}{\{} D_3 \underset{6}{\}} \underset{2}{\}} \underset{1}{\}}$	$\underset{1}{\{} D_0 \underset{3}{\{} D_2 \underset{3}{\}} \underset{2}{\{} \underset{2}{\}} \underset{1}{\}}$
$(A2)$	$\underset{1}{\{} D_0 \underset{4}{\{} D_1 \underset{4}{\}} \underset{3}{\{} \underset{5}{\{} \underset{5}{\}} \underset{3}{\}} \underset{2}{\{} \underset{6}{\{} D_3 \underset{6}{\}} \underset{2}{\}} \underset{1}{\}}$	$\underset{1}{\{} D_0 \underset{3}{\{} D_2 \underset{3}{\}} \underset{2}{\{} \underset{2}{\}} \underset{1}{\}}$
$(A3)$	$\underset{1}{\{} D_0 \underset{4}{\{} D_1 \underset{4}{\}} \underset{2}{\{} \underset{6}{\{} D_3 \underset{6}{\}} \underset{2}{\}} \underset{1}{\}} \quad r = 3$	$\underset{1}{\{} D_0 \underset{3}{\{} D_2 \underset{3}{\}} \underset{1}{\}} \quad r = 2$
$(A4)$	$\underset{1}{\{} D_0 \underset{4}{\{} D_1 \underset{4}{\}} \underset{2}{\{} D_3 \underset{2}{\}} \underset{1}{\}} \quad g = 2$	$\underset{1}{\{} D_0 \underset{3}{\{} D_2 \underset{3}{\}} \underset{1}{\}} \quad g = 6$
$(A5)$	$\underset{1}{\{} D_0 \underset{3}{\{} D_1 \underset{3}{\}} \underset{2}{\{} D_3 \underset{2}{\}} \underset{1}{\}}$	$\underset{1}{\{} D_0 \underset{2}{\{} D_2 \underset{2}{\}} \underset{1}{\}}$
$(A6)$	$\Rightarrow$ G 2	$\Rightarrow$ R 2

6.3 Analysis

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.

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

(P1) Each pair of brackets has at least one node residing in it.
(P2) There are at most n pairs of brackets, where n is the number of nodes of $\mathbf{D}(X)$.
(P3) The brackets are numbered with consecutive numbers $1, \ldots m$, where m is the number of bracket pairs.
(P4) 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 (P1)–(P4) remain true for level $l+1$.

(P1) 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.

(P2) is an immediate consequence of (P1) and the fact that (A2) removes all duplicate nodes.

(P3) This is the obvious result of (A5).

(P4) New brackets are added in (A1). Each added pair obtains the lowest unused number. Because of this and (P3), 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. □

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

Proof. (1) Suppose $\Rightarrow \mathrm{G}\,g$ appears on infinitely many levels. That means a pair of brackets with number g becomes green infinitely often. Suppose further that no $\Rightarrow \mathrm{R}\,r$ with $r \leq g$ appears after some level l_0. It means that the pair numbered g is neither removed nor renumbered in the subsequent levels. Let l_1 be the first level after l_0 where the pair g becomes green. Step (A4) removed all brackets nested with it. Step (A1) adds a pair of brackets around each node reached by a marked edge.

Thus, on level l_1+1 any node inside the pair g enclosed in additional brackets has been reached from level l_1 by a marked edge. Any node on level l_1+2 that is enclosed in additional brackets inside the pair g had either such brackets already on level l_1+1, or has been reached from that level by a marked edge. In any case, the node has been reached from level l_1 by a marked path. In general, any node on a subsequent level enclosed in additional brackets inside the pair g has been reached from level l_1 by a marked path.

The pair g becomes green again at some level l_2 when all nodes inside it are enclosed in additional brackets. Let $l_1, l_2, l_3, \ldots$ be the consecutive levels where the pair g became green. The above reasoning applies to all these levels: each node within the pair g on level l_i for $i > 1$ is reached by a marked path from some node on level l_{i-1}.

For each node on level l_i with $i > 1$ select one marked path leading to it from a node on level l_{i-1}. In addition, select one path from level 1 to each node on level l_1. The selected paths form an infinite tree with finite branching and root on level 1. By König's lemma, there exists in this tree an infinite path from the root. For each $i \geq 1$, each part of the path between levels l_i and l_{i-1} contains at least one marked edge. It is thus a live path.

(2) Suppose $\mathbf{G}(X,w)$ contains a live path p. Each level of the DAG contains a node belonging to p. We denote it generically by D_p (it is, in general, different for different levels).

Suppose the pair number 1 becomes green infinitely often. That means $\Rightarrow$ G 1 appears infinitely often and $\Rightarrow$ R 1 not at all: pair 1 is not becoming empty, and all pairs removed by (A3) have numbers greater than 1. The condition stated by the Theorem is satisfied.

Suppose now that pair 1 is no more turning green after some level l_1. Each time p reaches the next level via a marked edge, a new pair of brackets is added around D_p. The first such pair resides in the pair 1; let its number be m_1. As pair 1 does not turn green any more, the pair m_1 is not removed. Two things may happen:

(a) A pair with number $m < m_1$ becomes empty and is deleted by (A3). As the result, m_1 is reduced by (A5).

(b) The annotation contains other occurrences of D_p, such as D'_p or D''_p below:

$$\underset{1}{\{} \cdots \underset{m_1}{\{} \cdots D_p \cdots \underset{m_1}{\}} \cdots \underset{m}{\{} \cdots D'_p \cdots \underset{m}{\}} \cdots D''_p \cdots \underset{1}{\}}\,.$$

If the other occurrence resides in pair 1, such as D''_p above, it is removed by (A2) because the nesting pattern (1) is higher than (1 - m_1 - ...).

If the other occurrence does not reside in pair 1, it must be within some pair of brackets that resides in pair 1. If it is within the pair m_1, either this occurrence or the original one is removed by (A2), and we are left with D_p in the pair m_1.

If the other occurrence, such as D'_p above, is within another pair with some number m, we have two possibilities:

– If $m > m_1$, the occurrence is removed by (A2) because the nesting pattern $(1\text{-}m\text{-}\dots)$ is higher than $(1\text{-}m_1\text{-}\dots)$, and we are left with D_p in the pair m_1.
– If $m < m_1$, the original occurrence is removed by (A2) because the nesting pattern $(1\text{-}m_1\text{-}\dots)$ is higher than $(1\text{-}m\text{-}\dots)$, and we are left with D_p in a different pair residing in pair 1, with a lower number.

As the number on the brackets cannot be lowered indefinitely, (a) and (b) can only happen finitely many times, and do not occur any more after some level l_2. At any level $l > l_2$, D_p is enclosed in a pair residing in pair 1, with a number $m_1' \leq m_1$ that does not change.

Suppose the pair m_1' becomes green infinitely often. That means $\Rightarrow \mathrm{G}\, m_1'$ appears infinitely often. As neither the pair m_1', nor any pair numbered $m < m_1'$ become empty, $\Rightarrow \mathrm{R}\, r$ with $r \leq m1'$ does not appear after level l_2. The stated condition is satisfied.

Suppose now that the pair m_1' is turning green for the last time at some level l_3. Each time p reaches the next level via a marked edge, a new pair of brackets is added around D_p. The first such pair resides in the pair m_1'; let its number be m_2. As the pair m_1' no longer turns green, the pair m_2 is not removed, but may be renumbered by (A5) or replaced by a pair with lower number by (A2). As before, no renumbering occurs after some level l_4, with m_2 possibly reduced to m_2'. As before, either the pair m_2' is turning green infinitely often, or another pair of brackets residing in pair m_2' is added around D_p.

This step can be repeated, adding each time a new pair of brackets around D_p. However, according to (P2), the number of brackets cannot grow without a bound, so the process must end with some pair m_i' turning green infinitely often after no $\Rightarrow \mathrm{R}\, r$ with $r \leq m_i'$ appear any more. □

7 Towards a deterministic automaton

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

In fact, we do not need to construct $\mathbf{D}(X)$ and $\mathbf{G}(X, w)$; 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 (P2), 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 $D_0, \dots, D_n$. 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 $D_0, \dots, D_n$ 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 $\mathbf{G}(X, w)$. According to Proposition 2, $\mathbf{A}(X)$ accepts w if and only if $\mathbf{G}(X, w)$ contains a live path, and $\mathbf{G}(X, w)$ 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 $w \in X$. $\square$

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**
$A = \underset{1}{\{} D_0 \underset{1}{\}}$	$B = \underset{1}{\{} D_0 \underset{2}{\{} D_1 \underset{2}{\}} \underset{1}{\}}$	$A = \underset{1}{\{} D_0 \underset{1}{\}}$
$B = \underset{1}{\{} D_0 \underset{2}{\{} D_1 \underset{2}{\}} \underset{1}{\}}$	$C = \underset{1}{\{} D_0 \underset{3}{\{} D_1 \underset{3}{\}} \underset{2}{\{} D_3 \underset{2}{\}} \underset{1}{\}} \Rightarrow \mathrm{G2}$	$D = \underset{1}{\{} D_0 \underset{2}{\{} D_2 \underset{2}{\}} \underset{1}{\}}$
$C = \underset{1}{\{} D_0 \underset{3}{\{} D_1 \underset{3}{\}} \underset{2}{\{} D_3 \underset{2}{\}} \underset{1}{\}}$	$C = \underset{1}{\{} D_0 \underset{3}{\{} D_1 \underset{3}{\}} \underset{2}{\{} D_3 \underset{2}{\}} \underset{1}{\}} \Rightarrow \mathrm{G2}$	$D = \underset{1}{\{} D_0 \underset{2}{\{} D_2 \underset{2}{\}} \underset{1}{\}} \Rightarrow \mathrm{R2}$
$D = \underset{1}{\{} D_0 \underset{2}{\{} D_2 \underset{2}{\}} \underset{1}{\}}$	$E = \underset{1}{\{} D_0 \underset{3}{\{} D_1 \underset{3}{\}} \underset{2}{\{} D_4 \underset{2}{\}} \underset{1}{\}} \Rightarrow \mathrm{G2}$	$A = \underset{1}{\{} D_0 \underset{1}{\}} \Rightarrow \mathrm{R2}$
$E = \underset{1}{\{} D_0 \underset{3}{\{} D_1 \underset{3}{\}} \underset{2}{\{} D_4 \underset{2}{\}} \underset{1}{\}}$	$C = \underset{1}{\{} D_0 \underset{3}{\{} D_1 \underset{3}{\}} \underset{2}{\{} D_3 \underset{2}{\}} \underset{1}{\}} \Rightarrow \mathrm{R2}$	$D = \underset{1}{\{} D_0 \underset{2}{\{} D_2 \underset{2}{\}} \underset{1}{\}} \Rightarrow \mathrm{R3}$

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

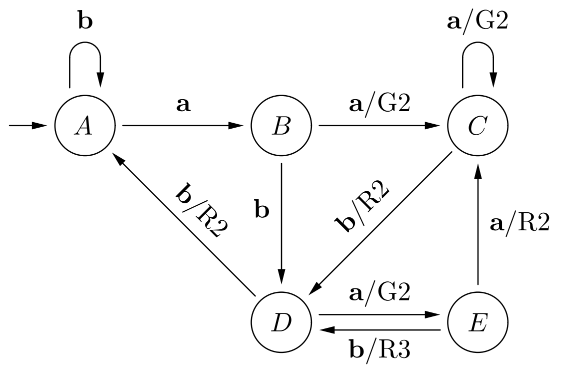

Fig. 4

8 Comments

By eliminating the ordering of nodes, the described construction gives fewer states than one in [5], but their number is still exponential. The construction and proofs have been simplified by introducing the derivative automaton and the run DAG.

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 DAG 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.

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?

References

1. Brzozowski, J.A.: Derivatives of regular expressions. Journal of the ACM **11**(4) (1964) 481–494
2. Owens, S., Reppy, J., 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.: The theory of general events and its application to parallel programming. Technical paper TP 18.220, IBM Nordic Laboratory, Lidingö, Sweden (1972)
5. Redziejowski, R.R.: Construction of a deterministic ω-automaton using derivatives. Informatique Théorique et Applications **33** (1999) 133–158
6. Safra, S.: On the complexity of ω-automata. In: Proc. 29th Annual Symposium on Foundations of Computer Science, IEEE (1988) 319–327
7. Safra, S.: Complexity of automata on infinite objects. Master's thesis, Weizmann Institute of Science, Rehovot, Israel (1989)