RAIRO-Theor. Inf. Appl. 45 (2011) 347–370
DOI: 10.1051/ita/2011107

Available online at:
www.rairo-ita.org

CONSTRUCTION OF TREE AUTOMATA FROM REGULAR EXPRESSIONS*

Dietrich Kuske[1] and Ingmar Meinecke[2]

Abstract. Since recognizable tree languages are closed under the rational operations, every regular tree expression denotes a recognizable tree language. We provide an alternative proof to this fact that results in smaller tree automata. To this aim, we transfer Antimirov's partial derivatives from regular word expressions to regular tree expressions. For an analysis of the size of the resulting automaton as well as for algorithmic improvements, we also transfer the methods of Champarnaud and Ziadi from words to trees.

Mathematics Subject Classification. 68Q45.

Introduction

One of the most prominent topics in formal language theory is the comparison of different finite descriptions for potentially infinite objects – the languages. The result of Kleene [14] states the equivalence between finite automata and regular expressions for languages of finite words. The transformation of a finite automaton into an equivalent regular expression is a prototypical example of dynamic programming. The converse transformation is of direct practical consequence *e.g.* in text processing. For this reason, several methods were proposed within the last decades to find more efficient algorithms, see [19,20] for surveys. For teaching purposes, one often uses an inductive construction. The most common construction is the standard or position automaton (Glushkov [10] and McNaughton and Yamada [17]). Brzozowski's construction [4] of a deterministic finite automaton

Keywords and phrases. Trees, automata, regular expressions, partial derivatives.

* *I. Meinecke was supported by the German Research Foundation (DFG) within the project DR 202/10-1.*

[1] Fachgebiet Theoretische Informatik, Technische Universität Ilmenau, Postfach 100565, 98684 Ilmenau, Germany. dietrich.kuske@tu-ilmenau.de

[2] Institut für Informatik, Universität Leipzig, PF 100920, 04009 Leipzig, Germany. meinecke@informatik.uni-leipzig.de

Article published by EDP Sciences

uses derivates of regular expressions. This approach was modified by Antimirov [2] who defined *partial derivatives* to construct a non-deterministic automaton from a regular expression.

Kleene's theorem was lifted to the setting of trees [22], also *cf.* [8,9], which are one of the most fundamental concepts in computer science. A regular tree expression defines a language of ordered trees. An inductive construction even produces a tree automaton accepting this language. The number of states of this automaton is exactly the number of iterations in the expression E plus $|E|_\Sigma$ where $|E|_\Sigma$ is the number of occurrences of symbols from the ranked alphabet in E. In this paper, we define partial derivatives for regular tree expressions and build by their help a non-deterministic finite tree automaton recognizing the language denoted by the regular expression. The concept of partial derivatives will yield a tree automaton with at most $|E|_\Sigma$ states and $|E|_\Sigma^2$ transitions. The construction of this tree automaton and the correctness proof is combined with algorithmic considerations to build this automaton. We adapt and modify the approach by Champarnaud and Ziadi [5,6] in the word case who extended work of Berry and Sethi [3]. Here, we use linearizations of regular tree expressions. The main idea is to distinguish occurrences of the same symbol at different positions in the regular expression. By doing so, we can ensure a certain uniqueness of the partial derivatives. As it turns out, the partial derivatives of the original regular expression are just projections of the partial derivatives of the linearized regular expression. This approach results in two main advantages: firstly, the desired automaton is in fact a quotient of an automaton that stems from the linearized regular expression. This way we also get the upper bound on the number of transitions mentioned above. Secondly, the theoretical results allow for an efficient algorithm working in the syntax-tree of E. We obtain an algorithm with $\mathcal{O}(R \cdot \mathrm{size}(E)^2)$ space and time complexity where R is the maximal rank of a symbol occuring in the finite ranked alphabet Σ and $\mathrm{size}(E)$ is the size of the regular expression.

Beside the standard and the partial derivative construction there are other proposals in the literature how to obtain an automaton from a regular expression. Especially, it would be interesting whether the construction of the follow automaton [7,12,13] carries over to the setting of trees. In this paper we consider ranked trees. However, regular expressions were explored for unranked trees in connection with XML. They are used in pattern matching, see *e.g.* [11]. In an extended abstract [15] of this paper, we wondered whether the concept of partial derivatives can lead to fruitful results and algorithms in this area. A first answer was given by Suzuki and Okui [21] who applied successfully the concept of partial derivatives to regular hedge expression patterns. Last but not least, Lombardy and Sakarovitch [16] applied the method of partial derivatives to a weighted setting for words. We are confident that this should be also possible for trees.

1. Trees, automata, and regular expressions

Let $\mathbb{N}$ be the set of non-negative integers. Throughout this paper, we fix a finite ranked alphabet $\Sigma = (\Sigma_m)_{m\in\mathbb{N}}$. The set T_Σ of trees over Σ is defined by the Backus-Naur form (BNF)

$$t ::= f(\underbrace{t, \ldots, t}_{m \text{ times}})$$

where $f \in \Sigma_m$. For the base case $c \in \Sigma_0$, we will write c instead of $c()$. A subset $L \subseteq T_\Sigma$ is called a *tree language*.

A (top-down) *tree automaton* over Σ is a tuple $\mathcal{A} = (Q, \Sigma, I, \Delta)$ where Q is a set of states, $I \subseteq Q$ is the set of initial states, and $\Delta = (\Delta_m)_{m\in\mathbb{N}}$ is the set of transitions[1] such that $\Delta_m \subseteq Q \times \Sigma_m \times Q^m$ for every $m \in \mathbb{N}$. Especially, $\Delta_0 \subseteq Q \times \Sigma_0$. A finite tree-automaton (or FTA) is a tree automaton $\mathcal{A}$ with only finitely many states and, thus, only finitely many transitions (note that there are only finitely many m with $\Sigma_m \neq \emptyset$).

As to whether a tree t is accepted by a tree automaton $\mathcal{A} = (Q, \Sigma, I, \Delta)$ is defined inductively along the construction of the tree t: if $t = c \in \Sigma_0$, then t is accepted by $\mathcal{A}$ iff there exists a state $q \in I$ with $(q, c) \in \Delta_0$. For $f \in \Sigma_m$ with $m > 0$, the tree $f(t_1, \ldots, t_m)$ is accepted by $\mathcal{A}$ iff there exist states $q \in I$ and $q_1, \ldots, q_m \in Q$ such that $(q, f, q_1, \ldots, q_m) \in \Delta_m$ and, for $1 \leq i \leq m$, the tree t_i is accepted by the tree automaton $(Q, \Sigma, \{q_i\}, \Delta)$. The language $L(\mathcal{A})$ recognized by $\mathcal{A}$ is the set of all trees t that are accepted by $\mathcal{A}$. A tree language L is *recognizable* if there is a FTA $\mathcal{A}$ with $L(\mathcal{A}) = L$.

We next introduce some constructions of tree languages that extend the rational operations on word languages. Let $f \in \Sigma_m$ and $L_1, \ldots, L_m \subseteq T_\Sigma$. Then we put

$$f(L_1, \ldots, L_m) = \{f(t_1, \ldots, t_m) \mid t_i \in L_i \text{ for } i = 1, \ldots, m\}.$$

For $L \subseteq T_\Sigma$ and $c \in \Sigma_0$ we define for every $t \in T_\Sigma$ inductively the non-uniform *substitution* $t[c \leftarrow L]$:

- $c[c \leftarrow L] = L$ and $d[c \leftarrow L] = \{d\}$ for every $d \in \Sigma_0$ with $d \neq c$;
- $f(t_1, \ldots, t_m)[c \leftarrow L] = f(t_1[c \leftarrow L], \ldots, t_m[c \leftarrow L])$.

Then the *c-product* of $L_1, L_2 \subseteq T_\Sigma$ is the language $L_1 \cdot_c L_2 = \bigcup_{t\in L_1} t[c \leftarrow L_2]$. Now the iterated c-products are defined for $L \subseteq T_\Sigma$ by

$$L^{0,c} = \{c\} \text{ and } L^{n+1,c} = L^{n,c} \cup L \cdot_c L^{n,c}.$$

The *c-iteration* of L is defined as $L^{*c} = \bigcup_{n\geq 0} L^{n,c}$.

It is well-known that a tree language L is recognizable if and only if it can be denoted by a regular expression. These *regular expressions* are defined by the

[1] In the term-rewriting terminology employed by [8], the transition $(q, f, q_1, \ldots, q_m)$ is denoted by the rule $q(f(x_1, \ldots, x_m)) \to f(q_1(x_1), \ldots, q_m(x_m))$.

following grammar in BNF[2]:

$$E ::= \emptyset \mid f(\underbrace{E, \ldots, E}_{m \text{ times}}) \mid (E + E) \mid (E \cdot_c E) \mid (E^{*c})$$

where $f \in \Sigma_m$ and $c \in \Sigma_0$. Again, we write c instead of $c()$ whenever $c \in \Sigma_0$.

The semantics $[\![E]\!]$ of a regular expression E is defined inductively by

$$\begin{aligned} [\![\emptyset]\!] &= \emptyset, & [\![f(E_1, \ldots, E_m)]\!] &= f([\![E_1]\!], \ldots, [\![E_m]\!]), \\ [\![(E + F)]\!] &= [\![E]\!] \cup [\![F]\!], & [\![(E \cdot_c F)]\!] &= [\![E]\!] \cdot_c [\![F]\!], \text{ and} \\ [\![(E^{*c})]\!] &= [\![E]\!]^{*c}. \end{aligned}$$

For a set M of regular expressions, we put $[\![M]\!] = \bigcup_{E \in M} [\![E]\!]$.

The set of all regular expressions over the ranked alphabet Σ is denoted by $\mathrm{EXP}(\Sigma)$. Let $|E|_f$ denote the number of occurrences of the letter $f \in \Sigma$ in E. The *alphabetic width* $|E|_\Sigma$ of E is defined by $|E|_\Sigma = \max\{1, \sum_{f \in \Sigma} |E|_f\}$, *i.e.*, $|E|_\Sigma$ is the number of occurrences of symbols from Σ in E but for technical reasons at least 1. Thus, $|\emptyset|_\Sigma = 1$. The *size* of E is defined inductively by: $\mathrm{size}(\emptyset) = \mathrm{size}(c) = 1$ for $c \in \Sigma_0$, $\mathrm{size}(f(E_1, \ldots, E_m)) = 1 + \sum_{i=1}^m \mathrm{size}(E_i)$, $\mathrm{size}\big((E + F)\big) = \mathrm{size}\big((E \cdot_c F)\big) = 1 + \mathrm{size}(E) + \mathrm{size}(F)$, and $\mathrm{size}\big((E^{*c})\big) = 1 + \mathrm{size}(E)$. Every regular expression E can be understood as a tree over the ranked alphabet $\Sigma \cup \{+, \cdot_c, {}^{*c}, \emptyset \mid c \in \Sigma_0\}$ where $+$ and $\cdot_c$ have rank 2, ^{*c} has rank 1, and $\emptyset$ has rank 0. This tree is called the *syntax-tree* t_E of E.

2. A direct construction

In this section, we will construct from a regular expression E a tree automaton $\mathcal{A}_E$ that accepts $[\![E]\!]$. The finiteness of this automaton will only be proved later. Our construction is based on partial derivates that we have to define and investigate first.

Let M be a set of regular expressions, F some regular expression, and $c \in \Sigma_0$. Then $M \cdot_c F$ denotes the set $\{(E \cdot_c F) \mid E \in M\}$. Similarly, we put for a set $\mathcal{M}$ of m-tuples of regular expressions

$$\mathcal{M} \cdot_c F = \big\{\big((E_1 \cdot_c F), (E_2 \cdot_c F), \ldots, (E_m \cdot_c F)\big) \mid (E_1, E_2, \ldots, E_m) \in \mathcal{M}\big\}.$$

Definition 2.1. For $g \in \Sigma_m$, $m \geq 1$, and a regular expression E, we define the sets $g^{-1}E$ of m-tuples of regular expressions inductively:

- $g^{-1}\emptyset = \emptyset$,

[2]For constructing an equivalent regular expression for a given tree automaton, it is necessary to introduce additional symbols of rank zero, *cf.* [9], Proposition 9.2. But since we are interested only in the converse, *i.e.*, the construction of automata from an expression, we do not have to differentiate between nullary symbols from the alphabet and additional ones.

- $g^{-1}\, f(E_1, E_2, \ldots, E_n) = \begin{cases} \{(E_1, E_2, \ldots, E_n)\} & \text{if } f = g \\ \emptyset & \text{if } f \neq g, \end{cases}$
- $g^{-1}(E + F) = g^{-1}E \cup g^{-1}F$,
- $g^{-1}(E \cdot_c F) = \begin{cases} (g^{-1}E) \cdot_c F \cup g^{-1}F & \text{if } c \in [\![E]\!] \\ (g^{-1}E) \cdot_c F & \text{otherwise,} \end{cases}$
- $g^{-1}(E^{*c}) = (g^{-1}E) \cdot_c (E^{*c})$.

Let $\Sigma_{\geq 1} = \bigcup_{m \geq 1} \Sigma_m = \Sigma \setminus \Sigma_0$ denote the set of non-nullary symbols from the ranked alphabet Σ. Following Antimirov, we define further functions ∂_w for finite words $w \in \Sigma_{\geq 1}^*$ over the alphabet $\Sigma_{\geq 1}$. By ε we denote the empty word.

Definition 2.2. Let E be a regular expression. Then $\partial_\varepsilon E = \{E\}$ and, for $w \in \Sigma_{\geq 1}^*$ and $g \in \Sigma_{\geq 1}$, the set $\partial_{wg}E$ consists of all regular expressions F that appear in some tuple from $g^{-1}E'$ for some $E' \in \partial_w E$. For a set of words $W \subseteq \Sigma_{\geq 1}^*$ and a regular expression E, we put $\partial_W E = \bigcup_{w \in W} \partial_w E$.

The function ∂_w is called the *partial derivative with respect to* w.

Note that $\partial_{wg}E = \partial_g \partial_w E = \bigcup_{E' \in \partial_w E} \partial_g E'$ for all $w \in \Sigma_{\geq 1}^*$ and $g \in \Sigma_{\geq 1}$. Further note that we consider derivatives with respect to words over the non-nullary symbols from Σ and not with regard to trees.

Example 2.3. Let $\Sigma_0 = \{a, b\}$, $\Sigma_1 = \{g, h\}$, and $\Sigma_2 = \{f\}$. Consider the regular tree expression

$$E = \Big(\underbrace{\Big(f\big(g\big(h(a)\big), g(b)\big)^{*a} \Big)}_{E_1} \cdot_b \underbrace{\big(h(a) + h(b)\big)}_{E_2} \Big).$$

Note that $b \notin [\![E_1]\!]$. Hence $\partial_g E = \partial_h E = \emptyset$ and

$$\partial_f E = \Big\{ \Big(\big(g\big(h(a)\big) \cdot_a E_1\big) \cdot_b E_2 \Big),\ \Big(\big(g(b) \cdot_a E_1\big) \cdot_b E_2 \Big) \Big\}.$$

Furthermore, we compute the following partial derivatives:

$$\begin{aligned} \partial_{fg}E &= \Big\{ \Big(\big(h(a) \cdot_a E_1\big) \cdot_b E_2 \Big),\ \Big(\big(b \cdot_a E_1\big) \cdot_b E_2 \Big) \Big\}, \\ \partial_{fgh}E &= \Big\{ \Big(\big(a \cdot_a E_1\big) \cdot_b E_2 \Big),\ a, b \Big\}; \end{aligned}$$

where the last equality is due to $\partial_h\big((b \cdot_a E_1) \cdot_b E_2\big) = \partial_h E_2 = \{a, b\}$. Notice that $\partial_{fghf}E = \partial_f E$. Hence, together with $\partial_\varepsilon E = \{E\}$ we obtain eight different partial derivatives of E.

A symbol $f \in \Sigma$ occurs *unguarded* in E if no ancestor in the syntax tree t_E is labeled by an element of Σ. We will be interested in the number $\langle E \rangle_f$ of unguarded occurrences of f in E that can be defined inductively by:

- $\langle \emptyset \rangle_f = 0$,

- $\langle f(E_1,\dots,E_m)\rangle_f = 1$ and $\langle g(E_1,\dots,E_n)\rangle_f = 0$ for $g \neq f$,
- $\langle (E_1 + E_2)\rangle_f = \langle (E_1 \cdot_c E_2)\rangle_f = \langle E_1\rangle_f + \langle E_2\rangle_f$, and
- $\langle (E^{*c})\rangle_f = \langle E\rangle_f$.

Proposition 2.4. *Let E be a regular expression and $g \in \Sigma_{\geq 1}$. Then $|g^{-1}E| \leq \langle E\rangle_g$. Especially, if $|E|_g = 0$ then $g^{-1}E = \partial_g E = \emptyset$.*

Proof. The claim is shown by induction on the construction of E: for $E = \emptyset$, the claim is trivial. If we have $E = f(E_1,\dots,E_m)$ and $f \neq g$, then $|g^{-1}E| = 0$, so the claim is also trivial. If $f = g$, then $|g^{-1}E| = 1 = \langle E\rangle_g$.

Next consider the case $(E + F)$: then

$$|g^{-1}(E+F)| \leq |g^{-1}E| + |g^{-1}F| \leq \langle E\rangle_g + \langle F\rangle_g = \langle (E+F)\rangle_g.$$

For the product, we have

$$\begin{aligned} |g^{-1}(E \cdot_c F)| &\leq |(g^{-1}E) \cdot_c F| + |g^{-1}F| \leq |g^{-1}E| + |g^{-1}F| \\ &\leq \langle E\rangle_g + \langle F\rangle_g = \langle (E \cdot_c F)\rangle_g. \end{aligned}$$

Similarly, we obtain for the iteration

$$|g^{-1}(E^{*c})| = |(g^{-1}E) \cdot_c (E^{*c})| = |g^{-1}E| \leq \langle E\rangle_g = \langle (E^{*c})\rangle_g. \qquad \square$$

Next, we express the semantics of a regular expression $[\![E]\!]$ in terms of the semantics of the tuples from $g^{-1}E$.

Proposition 2.5. *For any regular expression E, we have*

$$\begin{aligned} [\![E]\!] = &\bigcup \{g([\![G_1]\!],\dots,[\![G_m]\!]) \mid g \in \Sigma_{\geq 1}, (G_1,\dots,G_m) \in g^{-1}E\} \\ &\cup \{c \in \Sigma_0 \mid c \in [\![E]\!]\}. \end{aligned} \tag{1}$$

Proof. Let $[\![E]\!]_0 = [\![E]\!] \cap \Sigma_0$. The proof proceeds by induction on the construction of the regular expression E. For $E = \emptyset$, equation (1) holds. Next let $E = f(E_1,\dots,E_m)$ where $f \in \Sigma_m$ and $E_1,\dots,E_m$ are regular expressions. We put $\overrightarrow{G} = (G_1,\dots G_m)$. Then

$$\begin{aligned} [\![E]\!] &= f([\![E_1]\!],\dots,[\![E_m]\!]) \\ &= \bigcup \{f([\![G_1]\!],\dots,[\![G_m]\!]) \mid \overrightarrow{G} \in f^{-1}E\} \quad (\text{since } f^{-1}E = \{(E_1,\dots,E_m)\}) \\ &= \bigcup \{g([\![G_1]\!],\dots,[\![G_m]\!]) \mid g \in \Sigma, \overrightarrow{G} \in g^{-1}E\} \quad (\text{since } g^{-1}E = \emptyset \text{ for } f \neq g). \end{aligned}$$

The proof in case $E = (E_1 + E_2)$ is immediate and therefore omitted. Next let $E = (E_1 \cdot_c E_2)$. Then we have

$$\begin{aligned} [\![E]\!] &= [\![E_1]\!] \cdot_c [\![E_2]\!] \\ &= \Big(([\![E_1]\!] \setminus \{c\}) \cdot_c [\![E_2]\!]\Big) \cup \Big(([\![E_1]\!] \cap \{c\}) \cdot_c [\![E_2]\!]\Big). \end{aligned}$$

By the induction hypothesis, the first of these two sets equals

$$\begin{aligned}&\Big(\bigcup\big\{f([\![G_1]\!],\dots,[\![G_m]\!]) \mid f\in\Sigma,\overrightarrow{G}\in f^{-1}E_1\big\}\Big)\cdot_c [\![E_2]\!]\cup([\![E_1]\!]_0\setminus\{c\})\\ =&\bigcup\big\{f([\![(G_1\cdot_c E_2)]\!],\dots,[\![(G_m\cdot_c E_2)]\!]) \mid f\in\Sigma,\overrightarrow{G}\in f^{-1}E_1\big\}\cup([\![E_1]\!]_0\setminus\{c\})\\ =&\bigcup\big\{f([\![H_1]\!],\dots,[\![H_m]\!]) \mid f\in\Sigma,\overrightarrow{H}\in (f^{-1}E_1)\cdot_c E_2\big\}\cup([\![E_1]\!]_0\setminus\{c\}).\end{aligned}$$

If $c\in[\![E_1]\!]$, then the second of the two sets above equals

$$[\![E_2]\!]=\bigcup\big\{f([\![H_1]\!],\dots,[\![H_m]\!]) \mid f\in\Sigma,\overrightarrow{H}\in f^{-1}E_2\big\}\cup[\![E_2]\!]_0,$$

otherwise it is empty. Hence equation (1) holds for $E=(E_1\cdot_c E_2)$.

Finally, consider the regular expression (E^{*c}). If $f(t_1,\dots,t_m)\in[\![(E^{*c})]\!]=[\![E]\!]^{*c}$, then there exists $n\geq 0$ with $f(t_1,\dots,t_m)\in[\![E]\!]^{n+1,c}\setminus[\![E]\!]^{n,c}$. Hence there exists $s\in[\![E]\!]$ with $f(t_1,\dots,t_m)\in\{s\}\cdot_c[\![E]\!]^{n,c}$. Since $f(t_1,\dots,t_m)\notin[\![E]\!]^{n,c}$, the tree s is of the form $s=f(s_1,\dots,s_m)$. By the induction hypothesis, we find $(G_1,\dots,G_m)\in f^{-1}E$ with $s_i\in[\![G_i]\!]$. Hence we obtain

$$\begin{aligned}f(t_1,\dots,t_m)&\in\{s\}\cdot_c[\![(E^{*c})]\!]\\ &\subseteq f([\![G_1]\!],\dots,[\![G_m]\!])\cdot_c[\![(E^{*c})]\!]\\ &=f([\![(G_1\cdot_c(E^{*c}))]\!],\dots,[\![(G_m\cdot_c(E^{*c}))]\!]).\end{aligned}$$

Since the tuple $\big((G_i\cdot_c(E^{*c}))\big)_{1\leq i\leq m}=(G_i)_{1\leq i\leq m}\cdot_c(E^{*c})$ belongs to $f^{-1}(E^{*c})$, we showed the containment "$\subseteq$" of equation (1).

Conversely let $f\in\Sigma$ and $\overrightarrow{H}\in f^{-1}(E^{*c})=(f^{-1}E)\cdot_c(E^{*c})$. Then there exists a tuple of regular expressions $\overrightarrow{G}\in f^{-1}E$ with $\overrightarrow{H}=\overrightarrow{G}\cdot_c(E^{*c})$. Hence we get

$$\begin{aligned}f([\![H_1]\!],\dots,[\![H_m]\!])&=f([\![(G_1\cdot_c(E^{*c}))]\!],\dots,[\![(G_m\cdot_c(E^{*c}))]\!])\\ &=f([\![G_1]\!],\dots,[\![G_m]\!])\cdot_c(E^{*c}).\end{aligned}$$

By the induction hypothesis, $f([\![G_1]\!],\dots,[\![G_m]\!])\subseteq[\![E]\!]$, so we can continue

$$\subseteq[\![E]\!]\cdot_c[\![(E^{*c})]\!]\subseteq[\![(E^{*c})]\!].\qquad\square$$

Let E be a regular expression and let $Q_E=\partial_{\Sigma^*_{\geq 1}}E$. Then we define a set of transitions Δ_E as

$$\begin{aligned}&\big\{\big(F,f,G_1,G_2,\dots,G_m\big) \mid F\in Q_E,f\in\Sigma_m,m\geq 1,(G_1,\dots,G_m)\in f^{-1}F\big\}\\ \cup&\big\{(F,c) \mid F\in Q_E,c\in\Sigma_0,c\in[\![F]\!]\big\}.\end{aligned}$$

Furthermore, let $\mathcal{A}_E=(Q_E,\Sigma,\{E\},\Delta_E)$ denote the tree automaton whose only initial state is the regular expression E.

Theorem 2.6. *Let E be a regular expression over the ranked alphabet Σ. Then $\mathcal{A}_E$ is a tree automaton that accepts $[\![E]\!]$.*

Proof. We show by induction on the structure of trees that for any tree $t \in T_\Sigma$ and any regular expression F, the tree automaton $\mathcal{A}_F$ accepts t iff $t \in [\![F]\!]$.

First let $t = c \in \Sigma_0$. Now c is accepted by $\mathcal{A}_F$ iff there is a transition $(F, c) \in \Delta_F$. But this is the case iff $c \in [\![F]\!]$. Now let $t = f(s_1, \ldots, s_m)$ for some $m > 0$. Then t is accepted by $\mathcal{A}_F$ iff there is a transition $(F, f, (G_1, \ldots, G_m)) \in \Delta_F$ such that s_i is accepted by the tree automaton $(Q_F, \Sigma, \{G_i\}, \Delta_F)$ for all $1 \leq i \leq m$. Note that the reachable part of the automaton $(Q_F, \Sigma, \{G_i\}, \Delta_F)$ is the set of states Q_{G_i}. Hence, s_i is accepted by this automaton iff it is accepted by $\mathcal{A}_{G_i}$. By the induction hypothesis, this is equivalent to saying $s_i \in [\![G_i]\!]$. Since this holds for all $1 \leq i \leq m$, we have that t is accepted by $\mathcal{A}_F$ iff there exists $(G_1, \ldots, G_m) \in f^{-1}(F)$ with $s_i \in [\![G_i]\!]$ which is, by Proposition 2.5, equivalent to saying $t \in [\![F]\!]$. □

Example 2.7. Consider the regular expression

$$E = \Big(\underbrace{\big(f\big(g\big(h(a)\big), g(b)\big)^{*a} \big)}_{E_1} \cdot_b \underbrace{\big(h(a) + h(b)\big)}_{E_2} \Big)$$

from Example 2.3. There we computed the partial derivatives of E. Thus $Q_E = \{q_i \mid i = 0, \ldots, 7\}$ where

$$\begin{aligned}
&q_0 = E, && q_1 = \big(\big(g\big(h(a)\big) \cdot_a E_1\big) \cdot_b E_2\big), && q_2 = \big(\big(g(b) \cdot_a E_1\big) \cdot_b E_2\big),\\
&q_3 = \big(\big(h(a) \cdot_a E_1\big) \cdot_b E_2\big), && q_4 = \big(\big(b \cdot_a E_1\big) \cdot_b E_2\big), && q_5 = \big(\big(a \cdot_a E_1\big) \cdot_b E_2\big),\\
&q_6 = a, && q_7 = b.
\end{aligned}$$

The set of transitions Δ_E comprises

$$\begin{aligned}
&q_0 \xrightarrow{f} (q_1, q_2), && q_1 \xrightarrow{g} q_3, && q_2 \xrightarrow{g} q_4, && q_3 \xrightarrow{h} q_5, && q_4 \xrightarrow{h} q_6,\\
&q_4 \xrightarrow{h} q_7, && q_5 \xrightarrow{f} (q_1, q_2),\\
&q_0 \xrightarrow{a} \bot, && q_5 \xrightarrow{a} \bot, && q_6 \xrightarrow{a} \bot, && q_7 \xrightarrow{b} \bot.
\end{aligned}$$

Here, $q_0 \xrightarrow{f} (q_1, q_2)$ and $q_0 \xrightarrow{a} \bot$ mean that $(q_0, f, q_1, q_2), (q_0, a) \in \Delta_E$.

In the last example, the tree automaton resulting from our construction is finite. But so far, we did not prove in general that the tree automaton $\mathcal{A}_E$ has only finitely many states, *i.e.*, that $[\![E]\!]$ is recognizable. Theorem 3.16 will show that the number of states is linear and that the number of transitions is quadratic in the size of E. This will only be achieved after going through the following two constructions.

3. An indirect construction via linearizations

The idea of the indirect construction is as follows: in a regular expression E, uniquely mark the occurrences of letters from $\Sigma_{\geq 1}$. Then apply our direct construction to the resulting regular expression $\overline{E}$. The projection of this automaton accepts $[\![E]\!]$. As it turns out, a quotient of the automaton one obtains this way is isomorphic to the result of the direct construction.

3.1. Linear regular expressions

A regular expression E is *linear* if every letter $f \in \Sigma_{\geq 1}$ occurs at most once in E. Note that $c \in \Sigma_0$ may occur more than once. The following proposition is a consequence of Proposition 2.4.

Proposition 3.1. *Let E be a linear regular expression and $g \in \Sigma_m$ for $m \geq 1$. Then $|g^{-1}E| \leq 1$ and therefore $|\partial_g E| \leq m$.*

For $M \subseteq \mathrm{EXP}(\Sigma)$ and $g \in \Sigma_{\geq 1}$, we put $g^{-1}M = \bigcup\{g^{-1}E \mid E \in M\}$. Now we consider partial derivatives with respect to non-empty words for linear regular expressions.

Proposition 3.2. *Let E, F be linear regular expressions over the alphabet Σ such that also $(E + F)$ and $(E \cdot_c F)$ are linear. Let $w \in \Sigma^*_{\geq 1}$ and $g \in \Sigma_{\geq 1}$. Then the following hold true:*

- $g^{-1}\partial_w(E + F) = \begin{cases} g^{-1}\partial_w E & \text{if } |E|_g > 0, \\ g^{-1}\partial_w F & \text{otherwise.} \end{cases}$
- $g^{-1}\partial_w(E \cdot_c F) = \begin{cases} (g^{-1}\partial_w E) \cdot_c F & \text{if } |E|_g > 0, \\ \bigcup\{g^{-1}\partial_v F \mid \exists u \in \Sigma^*_{\geq 1} : w = uv \text{ and } c \in [\![\partial_u E]\!]\} & \\ \qquad\qquad \text{otherwise.} \end{cases}$
- *There are suffixes $v_1, \ldots, v_k$ of w such that*

$$g^{-1}\partial_w(E^{*c}) = \bigcup_{1 \leq i \leq k} (g^{-1}\partial_{v_i} E) \cdot_c (E^{*c}).$$

Proof. If $|E|_g = 0$, then $|\partial_w E|_g = 0$ implying $g^{-1}\partial_w E = \emptyset$ by Proposition 2.4. Now $g^{-1}\partial_w(E + F) = g^{-1}\partial_w E \cup g^{-1}\partial_w F$. If $|E|_g > 0$, then $g^{-1}\partial_w F = \emptyset$ and $g^{-1}\partial_w(E + F) = g^{-1}\partial_w E$. Otherwise we get $g^{-1}\partial_w E = \emptyset$ and $g^{-1}\partial_w(E + F) = g^{-1}\partial_w F$.

For the remaining claims we proceed by induction on $|w|$. The claims are obvious for $|w| = 0$. From now on let $w = \overline{w}f$ for some $\overline{w} \in \Sigma^*_{\geq 1}$ and $f \in \Sigma_{\geq 1}$. First, we consider $g^{-1}\partial_w(E \cdot_c F)$. By the induction hypothesis, we obtain

$$f^{-1}\partial_{\overline{w}}(E \cdot_c F) = \begin{cases} (f^{-1}\partial_{\overline{w}} E) \cdot_c F & \text{if } |E|_f > 0, \\ \bigcup\{f^{-1}\partial_v F \mid \exists u \in \Sigma^*_{\geq 1} : \overline{w} = uv \;\&\; c \in [\![\partial_u E]\!]\} & \text{otherwise.} \end{cases}$$

First, consider the case $|E|_f > 0$ and $|E|_g > 0$. Then $|F|_f = |F|_g = 0$ and therefore $g^{-1}F = \emptyset$. Hence we have $g^{-1}\partial_w(E \cdot_c F) = (g^{-1}\partial_w E) \cdot_c F$.

Next, let $|E|_f > |E|_g = 0$. Then (i) $g^{-1}\partial_w E = \emptyset$ and (ii) $\partial_v F = \emptyset$ for all non-empty suffixes v of w (since f occurs in v but not in F). Since, by the induction hypothesis, $f^{-1}\partial_{\overline{w}}(E \cdot_c F) = (f^{-1}\partial_{\overline{w}} E) \cdot_c F$, we get $\partial_w(E \cdot_c F) = \partial_f \partial_{\overline{w}}(E \cdot_c F) = (\partial_f \partial_{\overline{w}} E) \cdot_c F = (\partial_w E) \cdot_c F$ and therefore

$$\begin{aligned} g^{-1}\partial_w(E \cdot_c F) &= g^{-1}((\partial_w E) \cdot_c F) \\ &= \begin{cases} (g^{-1}\partial_w E) \cdot_c F & \text{if } c \notin [\![\partial_w E]\!] \\ (g^{-1}\partial_w E) \cdot_c F \cup g^{-1}F & \text{otherwise} \end{cases} \\ &\overset{(i)}{=} \begin{cases} \emptyset & \text{if } c \notin [\![\partial_w E]\!] \\ g^{-1}F & \text{otherwise} \end{cases} \\ &\overset{(ii)}{=} \bigcup\{g^{-1}\partial_v F \mid \exists u \in \Sigma^*_{\geq 1} : w = uv \;\&\; c \in [\![\partial_u E]\!]\} \end{aligned}$$

as required.

Now assume $|E|_f = 0$. Then the induction hypothesis implies $f^{-1}\partial_{\overline{w}}(E \cdot_c F) = \bigcup\{f^{-1}\partial_v F \mid \exists u \in \Sigma^*_{\geq 1} : \overline{w} = uv, c \in [\![\partial_u E]\!]\}$. Hence we obtain

$$\begin{aligned} \partial_w(E \cdot_c F) &= \partial_f \partial_{\overline{w}}(E \cdot_c F) \\ &= \bigcup\{\partial_f \partial_v F \mid \exists u \in \Sigma^*_{\geq 1} : \overline{w} = uv, c \in [\![\partial_u E]\!]\} \\ &= \bigcup\{\partial_v F \mid \exists u \in \Sigma^*_{\geq 1} : w = uv, v \neq \varepsilon, c \in [\![\partial_u E]\!]\} \\ &= \bigcup\{\partial_v F \mid \exists u \in \Sigma^*_{\geq 1} : w = uv, c \in [\![\partial_u E]\!]\} \end{aligned}$$

where the last equality holds since $\partial_w E = \emptyset$. Applying g^{-1} to this equation yields

$$g^{-1}\partial_w(E \cdot_c F) = \bigcup\{g^{-1}\partial_v F \mid \exists u : w = uv, c \in [\![\partial_u E]\!]\}.$$

If $|E|_g = 0$, this is precisely what we wanted to show. Otherwise, we obtain $|F|_g = 0$ and therefore $g^{-1}\partial_v F = \emptyset$ for all $v \in \Sigma^*_{\geq 1}$. Hence, in this case the last expression equals $\emptyset$. Since also $g^{-1}\partial_w E = \emptyset$ (due to the non-occurrence of f in E), this equals $(g^{-1}\partial_w E) \cdot_c F$ as required. This shows the claim for $(E \cdot_c F)$.

Now consider the regular expression (E^{*c}). By the induction hypothesis, there are suffixes $\overline{v}_1, \ldots, \overline{v}_k$ of $\overline{w}$ such that

$$\begin{aligned} f^{-1}\partial_{\overline{w}}(E^{*c}) &= \bigcup_{1 \leq i \leq k} (f^{-1}\partial_{\overline{v}_i} E) \cdot_c (E^{*c}) \\ \text{and} \quad \partial_w(E^{*c}) &= \bigcup_{1 \leq i \leq k} (\partial_{\overline{v}_i f} E) \cdot_c (E^{*c}). \end{aligned}$$

Hence, for $v_i = \overline{v}_i f$ $(1 \leq i \leq k)$ we have

$$g^{-1}\partial_w(E^{*c}) = g^{-1}\left(\bigcup_{1\leq i\leq k} (\partial_{v_i}E)\cdot_c (E^{*c})\right)$$
$$= \begin{cases} \bigcup_{1\leq i\leq k}(g^{-1}\partial_{v_i}E)\cdot_c (E^{*c}) & \text{if } c \notin [\![\partial_{v_i}E]\!] \text{ for all } 1\leq i\leq k,\\ \bigcup_{1\leq i\leq k}(g^{-1}\partial_{v_i}E)\cdot_c (E^{*c}) \cup g^{-1}(E^{*c}) & \text{otherwise.}\end{cases}$$

Since $g^{-1}(E^{*c}) = (g^{-1}E)\cdot_c (E^{*c}) = (g^{-1}\partial_\varepsilon E)\cdot_c (E^{*c})$, the set of tuples of regular expressions $g^{-1}\partial_w(E^{*c})$ is of the required form. □

Proposition 3.3. *Let E be a linear regular expression, $u, w \in \Sigma^*_{\geq 1}$, and $g \in \Sigma_{\geq 1}$. Then we have:*

(1) $|g^{-1}\partial_u E| \leq 1$,
(2) *if $g^{-1}\partial_u E \neq \emptyset$ and $g^{-1}\partial_w E \neq \emptyset$, then $g^{-1}\partial_u E = g^{-1}\partial_w E$.*

Proof. The proof is by induction on the structure of E.

For $E = \emptyset$ the claim is immediate. Now consider the case $E = f(E_1, \ldots, E_n)$. Since E is linear, there is at most one i with $|E_i|_g > 0$, if no such i exists, set $i = 1$. Then we have

$$g^{-1}\partial_u E = \begin{cases} g^{-1}\partial_{u'}\{E_1, \ldots, E_n\} = g^{-1}\partial_{u'}E_i & \text{if } u = fu',\\ \{(E_1, \ldots, E_n)\} & \text{if } u = \varepsilon \;\&\; f = g,\\ \emptyset & \text{otherwise,}\end{cases}$$

where the first case is due to $|E_j|_g = 0$ for $j \neq i$, and, similarly for $g^{-1}\partial_w E$. By induction hypothesis, we get immediately $|g^{-1}\partial_u E| \leq 1$.

Assume $g^{-1}\partial_u E \neq \emptyset$ and $g^{-1}\partial_w E \neq \emptyset$. If $f = g$ is the first letter of $u = fu'$, then $\emptyset \neq g^{-1}\partial_u E = g^{-1}\partial_{u'}E_i = f^{-1}\partial_{u'}E_i = \emptyset$ since E is linear, a contradiction. Hence, either $f \neq g$ for the first letter f of u or $f = g$ and u is empty. Since the analogous holds for w, we obtain $u = \varepsilon$ iff $w = \varepsilon$. Now the claim follows immediately from the induction hypothesis.

For $E = (E_1 + E_2)$ the claims are immediate by Proposition 3.2 and the induction hypothesis.

Let $E = (E_1 \cdot_c E_2)$. If $|E_1|_g > 0$, then by Proposition 3.2 $g^{-1}\partial_u E = (g^{-1}\partial_u E_1)\cdot_c E_2$ as well as $g^{-1}\partial_w E = (g^{-1}\partial_w E_1)\cdot_c E_2$. Hence, $|g^{-1}\partial_u E| \leq 1$ by induction hypothesis. If $g^{-1}\partial_u E$ and $g^{-1}\partial_w E$ are non-empty, so are the sets $g^{-1}\partial_u E_1$ and $g^{-1}\partial_w E_1$. Hence, by the induction hypothesis, the claim follows. Suppose now $|E_1|_g = 0$. Then $g^{-1}\partial_u E$ is a finite union of sets of the form $g^{-1}\partial_{u'}E_2$ where every u' is a suffix of u. The induction hypothesis implies that any two non-empty of them are equal, *i.e.*, $g^{-1}\partial_u E = g^{-1}\partial_{u'}E_2$ for some u'. Similarly, $g^{-1}\partial_w E = g^{-1}\partial_{w'}E_2$ for some word w'. Now both claims follow from the induction hypothesis.

A similar argument can be applied in case $E = (F^{*c})$ with $(g^{-1}\partial_{u'}F)\cdot_c (F^{*c})$ in place of $g^{-1}\partial_{u'}E_1$. □

An immediate consequence of the last proposition is

Corollary 3.4. *Let E be a linear regular expression, $u, w \in \Sigma_{\geq 1}^*$ and $g \in \Sigma_m$ with $m \geq 1$. Then*

(1) $|\partial_{ug} E| \leq m$,
(2) *if $\partial_{ug} E \neq \emptyset$ and $\partial_{wg} E \neq \emptyset$, then $\partial_{ug} E = \partial_{wg} E$.*

By Proposition 3.2 and Corollary 3.4 we conclude

Corollary 3.5. *For a linear regular expression E and $w \in \Sigma_{\geq 1}^+$ we have $\partial_w(E^{*c}) = (\partial_u E) \cdot_c (E^{*c})$ for some non-empty suffix u of w.*

Next, we bound the number of partial derivatives of a linear regular expression.

Proposition 3.6. *Let E be a linear regular expression. Then we have $|\partial_{\Sigma_{\geq 1}^+} E| \leq |E|_\Sigma - 1$ and $|\partial_{\Sigma_{\geq 1}^*} E| \leq |E|_\Sigma$.*

Proof. Note that $\partial_{\Sigma_{\geq 1}^*} E = \partial_{\Sigma_{\geq 1}^+} E \cup \{E\}$. We apply induction on E. Recall that $|\emptyset|_\Sigma = 1$. Then we have $|\partial_{\Sigma_{\geq 1}^+} \emptyset| = 0 = |\emptyset|_\Sigma - 1$. For $E = f(E_1, \ldots, E_n)$, $g \in \Sigma_{\geq 1}$ and $u \in \Sigma_{\geq 1}^*$, we get

$$\partial_{gu} E = \begin{cases} \partial_u \{E_1, \ldots, E_n\} & \text{if } g = f, \\ \emptyset & \text{if } g \neq f. \end{cases}$$

Hence, $|\partial_{\Sigma_{\geq 1}^+} E| \leq \sum_{i=1}^n |\partial_{\Sigma_{\geq 1}^*} E_i| \leq \sum_{i=1}^n |E_i|_\Sigma = |E|_\Sigma - 1$. For $E = (E_1 + E_2)$ we use Proposition 3.2 and the induction hypothesis and obtain the assumption. If $E = (E_1 \cdot_c E_2)$, then again by Proposition 3.2: $|\partial_{\Sigma_{\geq 1}^+} E| \leq |\partial_{\Sigma_{\geq 1}^+} E_1| + |\partial_{\Sigma_{\geq 1}^+} E_2| \leq |E_1|_\Sigma - 1 + |E_2|_\Sigma - 1 \leq |E|_\Sigma - 1$. Finally, we conclude by Corollary 3.5 $|\partial_{\Sigma_{\geq 1}^+} (E^{*c})| \leq |\partial_{\Sigma_{\geq 1}^+} E| \leq |E|_\Sigma - 1 = |(E^{*c})|_\Sigma - 1$. □

3.2. The projection construction

Recall that Theorem 2.6 provides a possibly infinite tree automaton $\mathcal{A}_E$ that accepts $[\![E]\!]$. Assuming E to be linear, we are now in the position to improve this result:

Corollary 3.7. *Let E be a linear regular expression over the ranked alphabet Σ. Then $\mathcal{A}_E$ is a finite tree automaton with at most $|E|_\Sigma$ states and at most $|E|_\Sigma \cdot |\Sigma|$ transitions that accepts $[\![E]\!]$.*

Proof. The equality $L(\mathcal{A}_E) = [\![E]\!]$ was shown in Theorem 2.6. Since the set of states of $\mathcal{A}_E$ equals $\partial_{\Sigma_{\geq 1}^*} E$, the finite tree automaton has at most $|E|_\Sigma$ states by Proposition 3.6. For $f \in \Sigma_{\geq 1}$ and $D \in Q_E$, there is at most one transition of the form $\big(D, f, (G_1, \ldots, G_m)\big)$ by Proposition 3.3(1), *i.e.*, there are at most $|E|_\Sigma \cdot |\Sigma_{\geq 1}|$ transitions whose label belongs to $\Sigma_{\geq 1}$. In addition, there can be $|Q_E \times \Sigma_0| \leq |E|_\Sigma \cdot |\Sigma_0|$ transitions of the form (D, c) with $c \in \Sigma_0$. □

Remark 3.8. A top-down FTA $\mathcal{A} = (Q, \Sigma, I, \Delta)$ is *deterministic* if I is a singleton and $(q, f, q_1, \ldots, q_m), (q, f, p_1, \ldots, p_m) \in \Delta$ imply $q_i = p_i$ for all $i \in \{1, \ldots, m\}$. Due to Proposition 3.3(1), we have even proved that for a *linear* regular expression E the FTA $\mathcal{A}_E$ is a deterministic top-down automaton which implies the number of transitions given in the corollary. For arbitrary regular expressions E, the FTA $\mathcal{A}_E$ is in general not deterministic.

Let Γ and Σ be two alphabets with $\Gamma_0 \subseteq \Sigma_0$. A mapping $\eta : \Gamma \to \Sigma$ with $\eta(\Gamma_m) \subseteq \Sigma_m$ for every $m \in \mathbb{N}$ and $\eta(c) = c$ for all $c \in \Sigma_0$ is called a *projection*. We can extend η naturally to $\eta : \mathrm{EXP}(\Gamma) \to \mathrm{EXP}(\Sigma)$ by:

- $\eta(\emptyset) = \emptyset$, $\eta(f(E_1, \ldots, E_m)) = \eta(f)(\eta(E_1), \ldots, \eta(E_m))$,
- $\eta\big((E + F)\big) = (\eta(E) + \eta(F))$, $\eta\big((E \cdot_c F)\big) = (\eta(E) \cdot_c \eta(F))$, and $\eta\big((E^{*c})\big) = (\eta(E)^{*c})$.

Definition 3.9. Let E and $\overline{E}$ be regular expressions over the ranked alphabets Σ and Γ, respectively, and let $\eta : \Gamma \to \Sigma$ be a projection. We say that $\overline{E}$ is a *refinement* of E with respect to the projection η if $\eta(\overline{E}) = E$.

$\overline{E}$ is called a *linearization* of E with respect to η if $\overline{E}$ is linear and a refinement of E with respect to η.

Example 3.10. A linearization of the regular expression E from Example 2.3 is

$$\overline{E} = \Big(\big(f_1\big(g_2\big(h_3(a)\big), g_4(b)\big)^{*a}\big) \cdot_b \big(h_5(a) + h_6(b)\big)\Big)$$

where $\Gamma = \{f_1, g_2, g_4, h_3, h_5, h_6, a, b\}$ and $\eta : \Gamma \to \Sigma$ is given by $\eta : f_1 \mapsto f$, $g_2, g_4 \mapsto g$, $h_3, h_5, h_6 \mapsto h$, $a \mapsto a$, $b \mapsto b$.

Note that due to $\eta(c) = c$ for $c \in \Sigma_0$ both the constants from Σ_0 and the operations $\cdot_c$ and ^{*c} remain unchanged. By abuse of notation, we denote also the two natural continuations of η to Γ^* and to T_Γ by η. The following lemma is easily shown:

Lemma 3.11. *Let E be a regular expression and $\overline{E}$ a refinement of E with respect to $\eta : \Gamma \to \Sigma$. Then $\eta([\![\overline{E}]\!]) = [\![E]\!]$.*

Let E be an arbitrary regular expression. Then one can construct a small finite tree automaton $\overline{\mathcal{A}}_E$ accepting $[\![E]\!]$ as follows: firstly, construct some linearization $\overline{E}$ of E with respect to $\eta : \Gamma \to \Sigma$ (we can assume that every symbol from Γ appears in $\overline{E}$ and therefore $|\Gamma| \leq |\overline{E}|_\Sigma = |E|_\Sigma$). Secondly, build the finite tree automaton $\mathcal{A}_{\overline{E}}$ which then has at most $|\overline{E}|_\Sigma = |E|_\Sigma$ states and at most $|\overline{E}|_\Sigma \cdot |\Gamma| \leq |E|_\Sigma{}^2$ transitions. Thirdly, replace the transitions $(\overline{F}, \overline{f}, (\overline{G}_1, \ldots, \overline{G}_m))$ of this automaton by $(\overline{F}, \eta(\overline{f}), (\overline{G}_1, \ldots, \overline{G}_m))$. Then, by Lemma 3.11, the following is immmediate:

Corollary 3.12. *Let E be a regular expression. Then $\overline{\mathcal{A}}_E$ is a finite tree automaton with at most $|E|_\Sigma$ states and at most $|E|_\Sigma{}^2$ transitions that accepts $[\![E]\!]$.*

3.3. The quotient construction

We will now collapse some of the states of the automaton $\overline{\mathcal{A}}_E$. The resulting automaton will turn out to be isomorphic to the automaton $\mathcal{A}_E$ from our first construction.

We define the following equivalence relation $\sim$ on $Q_{\overline{E}}$:

$$\overline{F} \sim \overline{H} :\iff \eta(\overline{F}) = \eta(\overline{H}).$$

We denote the equivalence class of $\overline{G} \in Q_{\overline{E}}$ by $[\overline{G}]$ and put $Q_{\overline{E}}/\!\sim = \{[\overline{G}] \mid G \in Q_{\overline{E}}\}$. Let $\overline{G}_i, \overline{H}_i \in Q_{\overline{E}}$ with $\overline{G}_i \sim \overline{H}_i$ for $i = 1, \dots, m$ and $\overline{f}_1, \overline{f}_2 \in \Gamma_{\geq 1}$ with $\eta(\overline{f}_1) = \eta(\overline{f}_2) = f$. Then $\overline{f}_1(\overline{G}_1, \dots, \overline{G}_m) \sim \overline{f}_2(\overline{H}_1, \dots, \overline{H}_m)$. Thus the following quotient FTA is well-defined: $\widetilde{\mathcal{A}}_E = \left(Q_{\overline{E}}/\!\sim, \Sigma, \{[\overline{E}]\}, \Delta'_E\right)$ where

$$\Delta'_E = \left\{ \left([\overline{F}], f, [\overline{G}_1], \dots, [\overline{G}_m]\right) \mid \left(\overline{F}, f, \overline{G}_1, \dots, \overline{G}_m\right) \in \overline{\Delta}_E \right\}.$$

We will show that the FTA $\widetilde{\mathcal{A}}_E$ is isomorphic to $\mathcal{A}_E$ and, thus, in particular accepts the language $[\![E]\!]$. Here, two FTA $\mathcal{A} = (Q, \Sigma, I, \Delta)$ and $\mathcal{A}' = (Q', \Sigma, I', \Delta')$ are isomorphic if there is a bijective mapping $\varphi : Q \to Q'$ with $q \in I$ iff $\varphi(q) \in I'$ and $(q, f, q_1, \dots, q_m) \in \Delta$ iff $(\varphi(q), f, \varphi(q_1), \dots, \varphi(q_m)) \in \Delta'$. Therefore, we have to clarify that $\eta(\overline{F}) \in Q_E = \partial_{\Sigma^*_{\geq 1}}(E)$ for every $\overline{F} \in Q_{\overline{E}}$. The following fundamental relation between the partial derivatives of E and of a refinement $\overline{E}$ is firstly shown for partial derivatives with respect to a single letter g by an induction on the construction of E:

Proposition 3.13. *Let E be a regular expression over the ranked alphabet Σ and $\overline{E}$ a refinement of E with respect to $\eta : \Gamma \to \Sigma$. Then we have for every $g \in \Sigma_{\geq 1}$*

$$g^{-1}E = \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta(\overline{g}^{-1}\overline{E}) \qquad \text{and} \qquad \partial_g E = \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta(\partial_{\overline{g}}\overline{E}).$$

Proof. We prove the first equation. From this the second one follows easily.

We proceed by induction on the construction of E. For $E = \emptyset$, the claim is immediate. Let $E = f(E_1, \dots, E_n)$. If $g \neq f$, then $g^{-1}E = \emptyset = \overline{g}^{-1}\overline{E}$ for every $\overline{g} \in \eta^{-1}(g)$. So let $g = f$. Now $f^{-1}E = (E_1, \dots, E_n)$ on the one hand. On the other hand, $\overline{E} = \overline{f}(\overline{E_1}, \dots, \overline{E_n})$ for some $\overline{f} \in \eta^{-1}(f)$ and some refinements $\overline{E_i}$ of E_i with respect to η. Hence $\eta(\overline{f}^{-1}\overline{E}) = (E_1, \dots, E_n)$. Since $h^{-1}\overline{E} = \emptyset$ for all $h \neq \overline{f}$, the equality follows for $E = f(E_1, \dots, E_n)$.

Now, let $E = (E_1 + E_2)$. Then there are refinements $\overline{E_1}$ and $\overline{E_2}$ of E_1 and E_2 with respect to η such that $\overline{E} = (\overline{E_1} + \overline{E_2})$. Now $g^{-1}E$ is as follows:

$$\begin{aligned} g^{-1}(E_1 + E_2) = g^{-1}E_1 \cup g^{-1}E_2 &= \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta(\overline{g}^{-1}\overline{E_1}) \cup \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta(\overline{g}^{-1}\overline{E_2}) \\ &= \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta\left(\overline{g}^{-1}\overline{E_1} \cup \overline{g}^{-1}\overline{E_2}\right) = \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta\left(\overline{g}^{-1}\overline{E}\right). \end{aligned}$$

We turn to $E = (E_1 \cdot_c E_2)$. Again, there are refinements $\overline{E_1}$ and $\overline{E_2}$ of E_1 and E_2 with respect to η such that $\overline{E} = (\overline{E_1} \cdot_c \overline{E_2})$. First suppose $c \notin [\![E_1]\!]$. Since $\eta^{-1}(c) = \{c\}$, Lemma 3.11 implies $c \notin [\![\overline{E_1}]\!]$. Hence, we have for this case:

$$\begin{aligned} g^{-1}(E_1 \cdot_c E_2) &= (g^{-1}E_1) \cdot_c E_2 \\ &= \left(\bigcup_{\overline{g} \in \eta^{-1}(g)} \eta(\overline{g}^{-1}\overline{E_1}) \right) \cdot_c E_2 \quad = \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta((\overline{g}^{-1}\overline{E_1}) \cdot_c \overline{E_2}) \\ &= \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta(\overline{g}^{-1}(\overline{E_1} \cdot_c \overline{E_2})) \quad = \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta(\overline{g}^{-1}\overline{E}). \end{aligned}$$

The case of $c \in [\![E_1]\!]$ is similar. For E^{*c} we conclude:

$$\begin{aligned} g^{-1}(E^{*c}) &= (g^{-1}E) \cdot_c E^{*c} \\ &= \left(\bigcup_{\overline{g} \in \eta^{-1}(g)} \eta(\overline{g}^{-1}\overline{E}) \right) \cdot_c E^{*c} \\ &= \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta((\overline{g}^{-1}\overline{E}) \cdot_c \overline{E}^{*c}) = \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta(\overline{g}^{-1}(\overline{E^{*c}})). \end{aligned}$$

□

Now we lift this result to partial derivatives with respect to arbitrary words.

Theorem 3.14. *Let E be a regular expression over the ranked alphabet Σ and $\overline{E}$ a refinement of E with respect to $\eta : \Gamma \to \Sigma$. Then we have for every $w \in \Sigma^*_{\geq 1}$*

$$\partial_w E = \bigcup_{\overline{w} \in \eta^{-1}(w)} \eta(\partial_{\overline{w}}\overline{E}).$$

Proof. We proceed by induction on the length of w where the case $w = \varepsilon$ is trivial. By Proposition 3.13, the assumption holds for $|w| = 1$. Now consider $w = ug$ with $u \in \Sigma^*_{\geq 1}$ and $g \in \Sigma_{\geq 1}$. Using the induction hypothesis we get:

$$\partial_{ug}E = \partial_g\partial_u E = \partial_g\Big(\bigcup_{\overline{u} \in \eta^{-1}(u)} \eta\big(\partial_{\overline{u}}\overline{E}\big)\Big) = \partial_g\, \eta \left(\bigcup_{\overline{u} \in \eta^{-1}(u)} \partial_{\overline{u}}\overline{E} \right). \qquad (\star)$$

Consider the set $\overline{H} = \bigcup\{\partial_{\overline{u}}\overline{E} \mid \overline{u} \in \eta^{-1}(u)\}$. Then there is for every $F \in \eta(\overline{H})$ a refinement $\overline{F} \in \overline{H}$ such that, by Proposition 3.13,

$$\partial_g F = \bigcup_{\overline{g} \in \eta^{-1}(g)} \partial_{\overline{g}}\overline{F}.$$

Moreover, every $\overline{F} \in \overline{H}$ is a refinement of some $F \in \eta(\overline{H})$ with respect to η. Hence, we can continue equation ($\star$):

$$= \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta\Big(\partial_{\overline{g}}\big(\bigcup_{\overline{u} \in \eta^{-1}(u)} \partial_{\overline{u}}\overline{E}\big)\Big) = \bigcup_{\overline{g} \in \eta^{-1}(g)} \eta\Big(\bigcup_{\overline{u} \in \eta^{-1}(u)} \partial_{\overline{ug}}\overline{E}\Big)$$
$$= \bigcup_{\overline{ug} \in \eta^{-1}(ug)} \eta(\partial_{\overline{ug}}\overline{E}). \qquad \square$$

This shows the claim for $w = ug$.

Now we can identify the result of the quotient construction.

Theorem 3.15. *The finite tree automaton $\widetilde{\mathcal{A}}_E$ is isomorphic to $\mathcal{A}_E$.*

Proof. The isomorphism is given by $\varphi : Q_{\overline{E}}/\!\sim \to Q_E : [\overline{G}] \mapsto \eta(\overline{G})$. Firstly, φ really maps into Q_E. Indeed, $\overline{G} = \partial_{\overline{w}}\overline{E}$ for some $\overline{w}$. By Theorem 3.14, $\eta(\overline{G}) \in \partial_{\Sigma^*_{\geq 1}} E = Q_E$. Injectivity of φ is obvious by the definition of $\sim$. Surjectivity follows from Theorem 3.14. Certainly, $\varphi([\overline{E}]) = E$. Now, suppose $\big([\overline{F}], f, [\overline{G}_1], \ldots, [\overline{G}_m]\big) \in \Delta'_E$. Then there is a $\overline{f}$ such that $\big(\overline{F}, \overline{f}, \overline{G}_1, \ldots, \overline{G}_m\big) \in \Delta_{\overline{E}}$ which means $(\overline{G}_1, \ldots, \overline{G}_m) \in \overline{f}^{\,-1}\overline{F}$. But due to Proposition 3.13 we have $(G_1, \ldots, G_m) \in f^{-1}F$ where $G_i = \eta(\overline{G}_i)$ and $F = \eta(\overline{F})$. Vice versa, if now $(G_1, \ldots, G_m) \in f^{-1}F$, then there is an $\overline{f} \in \Gamma_{\geq 1}$ with $(\overline{G}_1, \ldots \overline{G}_m) \in \overline{f}^{\,-1}\overline{F}$. Moreover, we have for $c \in \Sigma_0$:

$$([\overline{F}], c) \in \Delta'_E \iff c \in [\![\overline{F}]\!] \iff c \in [\![\eta(\overline{F})]\!] \iff (\eta(\overline{F}), c) \in \Delta_E. \qquad \square$$

Now we will show that the FTA $\mathcal{A}_E$ from Theorem 2.6 is finite. The number of transitions of $\mathcal{A}_E$ is obviously bounded from above by $|Q_E| \cdot |\Sigma| \cdot |Q_E|^R \leq |E|_\Sigma{}^{R+1} \cdot |\Sigma|$ where R is the maximal rank appearing in Σ. However, as we will show next, the upper bound for the number of transitions is much smaller.

Theorem 3.16. *Let E be a regular expression. Then $\mathcal{A}_E$ is a finite tree automaton with at most $|E|_\Sigma$ states and at most $|E|_\Sigma{}^2$ transitions that accepts $[\![E]\!]$.*

Proof. The equality $L(\mathcal{A}_E) = [\![E]\!]$ was shown in Theorem 2.6. The numbers of states and transitions of $\mathcal{A}_E$ equal those of $\widetilde{\mathcal{A}}_E$ by Theorem 3.15. Since $\widetilde{\mathcal{A}}_E$ is a quotient of $\overline{\mathcal{A}}_E$, the result follows from the estimates in Corollary 3.12. $\square$

Compare this to the inductive construction of a finite tree automaton accepting $[\![E]\!]$: for union and c-product, one takes the disjoint union of the argument automata and adds some transitions. For c-iteration, one has to add one new state in order to accept the tree c. Hence, the inductive construction yields a finite tree automaton whose number of states equals $|E|_\Sigma$ plus the number of c-iterations applied in the construction. The number of transitions of that automaton is very difficult to analyse.

4. Algorithmic issues

Due to Theorem 3.15, we can construct the FTA $\widetilde{\mathcal{A}}_E$ to get the automaton $\mathcal{A}_E$. Following this line, Champarnaud and Ziadi [5] gave in the case of words an algorithm with an $\mathcal{O}(|E|_\Sigma \cdot \mathrm{size}(E)^2)$ space and time complexity. By algorithmic refinements they enhanced the algorithm to one with an $\mathcal{O}(\mathrm{size}(E)^2)$ space and time complexity. We can mainly adapt this algorithm for the construction of the FTA $\mathcal{A}_E$ from a regular tree expression E. Since the algorithm is based on a more detailed analysis of the structure of partial derivatives, we first prove some more facts about them.

4.1. Existence and form of partial derivatives

It follows immediately from Proposition 3.2, Corollary 3.5, and Theorem 3.14 that every partial derivative D is a product of sub-expressions of E. But we can characterize the form of the partial derivatives from $\Delta_f E := \bigcup\{\partial_{wf} E \mid w \in \Sigma_{\geq 1}^*\}$ for any linear expression E and $f \in \Sigma_{\geq 1}$ in even more detail.

The partial derivatives of a linear expression can be computed by following the path from the respective symbol f at position p up to the root of the syntax tree. The computation starts with the sub-expression E_{pk} of the kth son of f. We multiply sub-expressions either when we pass a $\cdot_c$-node from the left or when we pass a $*c$-node. When passing a $\cdot_c$-node from the right, we have to check whether we can reach a c to the left of the $\cdot_c$-node by cancellation of letters of rank greater than or equal to one. If this is the case, we just pass without any changes, if not, then there is no partial derivative with respect to f. In the following, we would like to put this more formal.

Positions p of a tree are defined as usual as finite words over $\mathbb{N}$ where ε is the position of the root. The positions of a tree are ordered by the prefix ordering $\leq$. Positions of an expression E are understood as those in the syntax-tree t_E. Every position p determines the *sub-expression of E at position p*, which is denoted by E_p. We refer to the label at position p in the syntax tree t_E by $\ell(p)$.

To decide the existence of partial derivatives we compute inductively the function κ with $\kappa(p) = \{c \in \Sigma_0 \mid \exists t \in [\![E_p]\!] : c \text{ occurs in } t\}$:

$$\kappa(p) = \begin{cases} \emptyset & \text{if } \ell(p) = \emptyset, \\ \{\ell(p)\} & \text{if } \ell(p) \in \Sigma_0, \\ \bigcup_{i=0}^{m-1} \kappa(pi) & \text{if } \ell(p) \in \Sigma_m \text{ with } m \geq 1, \\ \kappa(p0) \cup \kappa(p1) & \text{if } \ell(p) = +, \\ \kappa(p0) \setminus \{c\} \cup \kappa(p1) & \text{if } \ell(p) = \cdot_c \text{ and } c \in \kappa(p0), \\ \kappa(p0) & \text{if } \ell(p) = \cdot_c \text{ and } c \notin \kappa(p0), \\ \kappa(p0) \cup \{c\} & \text{if } \ell(p) = *c. \end{cases}$$

Lemma 4.1. *Let $E \in \mathrm{EXP}(\Sigma)$ be linear such that $f \in \Sigma_m$ with $m \geq 1$ occurs in E. Let p be the unique position with $\ell(p) = f$. Then $\Delta_f E \neq \emptyset$ if and only if for all positions q with $q1 \leq p$ and $\ell(q) = \cdot_c$ for some $c \in \Sigma_0$ we have $c \in \kappa(q0)$.*

Proof. The proof is by induction on E. For $E = \emptyset$, there can be no position with label f. If $E = f(E_1, \ldots, E_r)$, then $\Delta_f E \neq \emptyset$ and the condition is satisfied vacuously. Let $E = g(E_1, \ldots, E_m)$ with $g \neq f$. Then $\Delta_f E \neq \emptyset$ if and only if there is an $i \in \{1, \ldots, m\}$ with $\Delta_f E_i \neq \emptyset$. By induction hypothesis, we get the claim. Similarly, we proceed for $E = (E_0 + E_1)$. Now let $E = (E_0 \cdot_c E_1)$. If f occurs in E_0, then $\Delta_f E \neq \emptyset$ if and only if $\Delta_f E_0 \neq \emptyset$. The induction hypothesis for E_0 yields the claim for E. If f is contained within E_1, then $\Delta_f E \neq \emptyset$ if and only if $\Delta_f E_1 \neq \emptyset$ and there is a word $w \in \Sigma^*_{\geq 1}$ such that there is an $F \in \partial_w E_0$ with $c \in [\![F]\!]$, *cf.* Proposition 3.2 and Corollary 3.4. The last condition is satisfied if and only if $c \in \kappa(0)$ (which can be shown by an induction on the structure of E_0). Hence, the claim follows also for this case. Finally, let $E = (E_0^{*c})$. Then $\Delta_f E \neq \emptyset$ if and only if $\Delta_f E_0 \neq \emptyset$ and we apply the induction hypothesis. □

For technical reasons, we introduce an auxiliary symbol ϵ. For positions $p = p_1 p_2 \ldots p_n$ with $p_i \in \mathbb{N}$ and $n \geq 1$ we define the functions ι and σ by

$$\iota(p) = \begin{cases} \epsilon & \text{if } \ell(p_1 \ldots p_{n-1}) \in \Sigma \cup \{+\} \text{ or if } \ell(p_1 \ldots p_{n-1}) = \cdot_c \text{ and } p_n = 1, \\ c & \text{if } \ell(p_1 \ldots p_{n-1}) = *c \text{ or if } \ell(p_1 \ldots p_{n-1}) = \cdot_c \text{ and } p_n = 0, \end{cases}$$

and

$$\sigma(p) = \begin{cases} \epsilon & \text{if } \ell(p_1 \ldots p_{n-1}) \in \Sigma \cup \{+\} \text{ or} \\ & \text{if } \ell(p_1 \ldots p_{n-1}) = \cdot_c \text{ and } p_n = 1, \\ p_1 \ldots p_{n-1}1 & \text{if } \ell(p_1 \ldots p_{n-1}) = \cdot_c \text{ and } p_n = 0, \\ p_1 \ldots p_{n-1} & \text{if } \ell(p_1 \ldots p_{n-1}) = *c. \end{cases}$$

Note that both ι and σ at position $p = p_1 \ldots p_n$ depend on the label $\ell(p_1 \ldots p_{n-1})$ of the upper neighbour of p (which cannot be $\emptyset$). For a fixed position $p = p_1 \ldots p_n$, we abbreviate $\iota(p_1 \ldots p_i)$ and $\sigma(p_1 \ldots p_i)$ by $\iota_p(i)$ and $\sigma_p(i)$, respectively, for all prefixes $p_1 \ldots p_i$ of p.

Theorem 4.2. *Let $E \in \mathrm{EXP}(\Sigma)$ be linear with $f \in \Sigma_m$ ($m \geq 1$) occuring in E. Let $p = p_1 p_2 \ldots p_n$ ($p_i \in \mathbb{N}$) be the unique position of f in the syntax tree t_E. Let $i_1 < \ldots < i_r \leq n$ be precisely the indices for which $\iota_p(i_j) \neq \epsilon$ ($j = 1, \ldots, r$). We put for every $k \in \{0, \ldots, m-1\}$*

$$D_f(E)_k = \Big(\Big(\ldots \Big(E_{pk} \cdot_{\iota_p(i_m)} E_{\sigma_p(i_m)} \Big) \cdot_{\iota_p(i_{m-1})} \cdots \Big) \cdot_{\iota_p(i_1)} E_{\sigma_p(i_1)} \Big). \tag{2}$$

If $\Delta_f E \neq \emptyset$, then $\Delta_f E = \{D_f(E)_k \mid k = 0, \ldots, m-1\}$.

Especially, the number of factors in $D_f(E)_k$ is bounded by the number of products $\cdot_c$ and stars $*c$ appearing on the path from f to the root.

Proof. We proceed by induction on the structure of E. For $E = \emptyset$, the symbol f cannot occur in E. Let $E = g(E_1, \ldots, E_l)$. If $g = f$ (and $l = m$), then $\Delta_f E = \{E_1, \ldots, E_m\}$ which yields the assumption. Now let $g \neq f$. Then $\Delta_f E = \Delta_f E_i$ for some $i \in \{1, \ldots, l\}$. By induction hypothesis, $\Delta_f E_i = \{D_{E_i}(f)_k \mid k = 0, \ldots, m-1\}$. Note that $\iota(1) = \epsilon$. Hence, $\Delta_f E = \Delta_f E_i = \{D_f(E)_k \mid k = 0, \ldots, m-1\}$.

For $E = (E_0 + E_1)$, $\Delta_f E = \Delta_f E_i$ for either $i = 0$ or $i = 1$. Again by induction hypothesis, $\Delta_f E_i = \{D_f(E_i)_k \mid k = 0, \ldots, m-1\}$, $\iota(1) = \epsilon$, and $D_f(E)_k = D_f(E_i)_k$ which yields the assumption.

Now let $E = (E_0 \cdot_c E_1)$. First, assume that f occurs in E_0. Then $\Delta_f E = (\Delta_f E_0) \cdot_c E_1$. Here, $D_f(E)_k = D_f(E_0)_k \cdot_{\iota(1)} E_{\sigma(1)} = D_f(E_0)_k \cdot_c E_1$. Applying the induction hypothesis, we get $\Delta_f E = \{D_f(E)_k \mid k = 0, \ldots, m-1\}$. Secondly, let us suppose that f occurs in E_1. Since $\Delta_f E \neq \emptyset$, we have $\Delta_f E = \Delta_f E_1$. Moreover, $\iota(1) = \epsilon$. Hence, $D_f(E)_k = D_f(E_1)_k$ which yields the assumption.

Finally, let $E = (E_0^{*c})$ and, thus, $\Delta_f E = (\Delta_f E_0) \cdot_c (E_0^{*c})$. Here, we have $D_f(E)_k = D_f(E_0)_k \cdot_{\iota(1)} E_{\sigma(1)} = D_f(E_0)_k \cdot_c E$. Together with the induction hypothesis this shows the claim for the last case. □

4.2. An algorithm for computing the automaton

We give a sketch of the algorithm which follows mainly the lines of the respective algorithm for word expressions [5] but makes more use of symbolic computations.

Note that the syntax-tree of a linearization $\overline{E}$ is obtained from t_E by labelling each $g \in \Sigma_{\geq 1}$ additionally with its position. In the sequel, we will mainly work within this syntax-tree.

Recall that the partial derivatives of E are projections of the partial derivatives of the linearization $\overline{E}$, *cf.* Theorem 3.14. Moreover, due to Proposition 3.3 the non-empty partial derivatives $\partial_{\overline{wg}}\overline{E}$ depend just on the last symbol $\overline{g}$, *i.e.*, the unique position in the syntax-tree labelled by $\overline{g}$. We will calculate the partial derivatives of $\overline{E}$ as suggested by Theorem 4.2. Moreover, we compute the Σ_0-semantics (*i.e.*, the constants contained in the semantics of the expression) of the partial derivatives and their so-called FIRST-sets which will allow us to determine the transitions of the automaton afterwards. Formally, the set $\mathrm{FIRST}(E)$ of a regular tree expression E is the set of all $f \in \Sigma_{\geq 1}$ such that $f^{-1}E \neq \emptyset$.

Some preliminary computations

Here, we compute some entities needed in the sequel, namely

- the sub-expressions E_p of E for every position p;
- the sets $\Sigma_0 \cap [\![E_p]\!]$;
- the functions κ, ι, and σ from pages 363 and 364.

Using κ and Lemma 4.1 we propagate the information for which symbols f partial derivatives $\Delta_f \overline{E}$ exist to the nodes in the syntax tree. These steps can be done in $\mathcal{O}(\mathrm{size}(E)^2)$ space and time complexity.

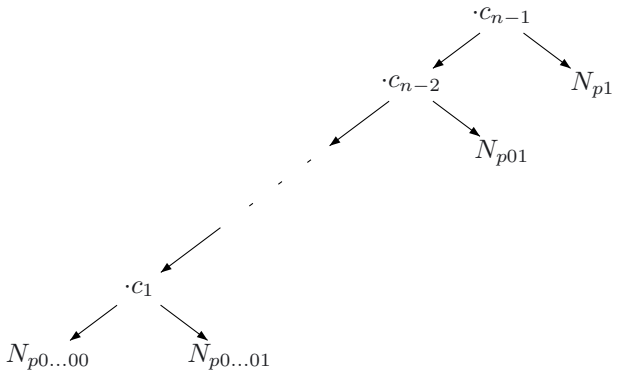

FIGURE 1. Representation S_p of the sub-expression E_p.

Concernig the sub-expressions E_p of E, we do not always need the complete sub-expression but just its maximal left-aligned factorization, *i.e.*, every sub-expression E_p is computed in the following form:

$$E_p = ((\ldots(F_1 \cdot_{c_1} F_2) \cdot_{c_2} \ldots) \cdot_{c_{n-1}} F_n)$$

where the F_i $(i = 1, \ldots, n)$ are sub-expressions of E_p and where F_1 is not a product. Whenever $n = 1$, we do compute E_p. Otherwise $(n > 1)$ we refer to the individual F_i in the above factorization just by their position.

Furthermore, we compare all sub-expressions by a lexicographic ordering. We refer to identical sub-expressions E_p and E_q by a common name N_p. This results in a representation S_p of every sub-expression E_p as follows

$$S_p = \big(\big(\ldots\big(N_{p0\ldots00} \cdot_{c_1} N_{p0\ldots01}\big) \cdot_{c_2} \ldots\big) \cdot_{c_{n-1}} N_{p1}\big),$$

cf. Figure 1.

Due to [1,18] a list of n strings of size $\mathcal{O}(t)$ over an alphabet of size k can be sorted lexicographically with complexity $\mathcal{O}(nt+k)$. Comparing consecutive strings in this list can be done within $\mathcal{O}(nt)$ complexity. Hence, the procedure takes an $\mathcal{O}(\text{size}(E)^2)$ space and time complexity[3].

Computing the states

We proceed by traversing the syntax-tree t_E from the leaves to the root and compute the partial derivatives of $\overline{E}$ according to Lemma 4.1 and Theorem 4.2. Hereby, we determine the partial derivatives symbolically, *i.e.*, using for the factors

[3]Here and in the sequel, we assume a RAM (random access machine) with a uniform cost measure as the basic machine model, *cf.* [23].

the pre-processed names of the sub-expressions. For the first factor we plug in the symbolic representation. Hence, we do compute instead of the expression in equation (2) the following

$$S_f(E)_k = \Big(\Big(\dots\Big(S_{pk} \cdot_{\iota(i_m)} N_{\sigma(i_m)}\Big) \cdot_{\iota(i_{m-1})} \dots\Big) \cdot_{\iota(i_1)} N_{\sigma(i_1)}\Big). \tag{3}$$

At every position p we do the following:

- we start the computation of new symbolic partial derivatives $S_f(E)$ whenever the label at p is $f \in \Sigma_{\geq 1}$ and $\Delta_f E \neq \emptyset$ (an information we have already processed);
- we update the symbolic partial derivatives $S_g(E)$ computed so far according to Theorem 4.2, thereby using the functions ι and σ.

At the root of t_E, we have also to add the representation S_ε of the whole expression as a symbolic partial derivative.

Since the number of factors in a partial derivative is bounded by the height of the syntax tree and since there are at most $|E|_\Sigma$ partial derivatives, we get an $\mathcal{O}(\text{size}(E) \cdot |E|_\Sigma)$ space complexity. The time complexity is $\mathcal{O}(\text{size}(E)^2)$ because we traverse the whole syntax tree and have to update at every position all symbolic partial derivatives computed so far.

Note that the maximal left-aligned factorization of every partial derivative $D_f(E)_k$ is almost given by equation (2). Only E_{pk} has to be replaced by its maximal left-aligned factorization. But this was already done at the symbolical level in the definition of $S_f(E)_k$, *cf.* equation (3). Hence,

$$D_f(E)_k = D_g(E)_\ell \iff S_f(E)_k = S_g(E)_\ell$$

for all $f, g \in \Sigma_{\geq 1}$ and respective $k, \ell \in \mathbb{N}$. Therefore, we just compare and identify the symbolic partial derivatives $S_f(E)_k$ to obtain the states of the automaton. Again, this is done by a lexicographic ordering with an $\mathcal{O}(\text{size}(E)^2)$ space and time complexity.

Computing the transitions for $c \in \Sigma_0$

We denote by D_p the partial derivative which belongs to position p. Now we calculate $\Sigma_0 \cap [\![D_p]\!]$ from the sets $\Sigma_0 \cap [\![E_q]\!]$ and obtain the respective transitions. This step has an $\mathcal{O}(|\Sigma_0| \cdot |E|_\Sigma)$ space and $\mathcal{O}(\text{size}(E) \cdot |E|_\Sigma)$ time complexity.

Computing the transitions for $f \in \Sigma_{\geq 1}$

Here, we have to compute the sets $\text{FIRST}(E_p)$ and $\text{FIRST}(D_p)$ for all positions p. Then we get for every D_p and $f_q \in \text{FIRST}(D_p)$ of rank $m+1$ (where q is the position where f occurs) a transition

$$D_p \xrightarrow{f_q} (D_{q0}, D_{q1}, \dots, D_{qm}).$$

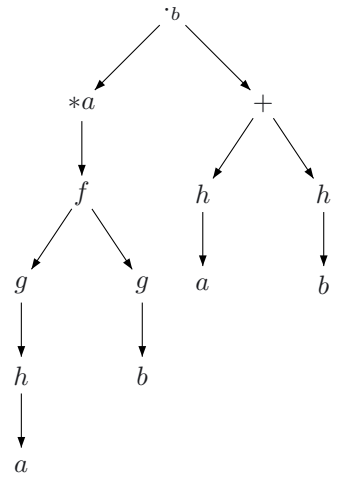

FIGURE 2. Syntax-tree t_E of E.

A projection onto the identified states and the alphabet Σ provides the transitions of the automaton.

The sets $\mathrm{FIRST}(E_p)$ can be computed with an $\mathcal{O}(\mathrm{size}(E))$ space and time complexity [5,24]. Then the computation of $\mathrm{FIRST}(D_p)$ takes an $\mathcal{O}(|E|_\Sigma{}^2)$ space and an $\mathcal{O}(|E|_\Sigma \cdot \mathrm{size}(E))$ time complexity [5,24]. Due to Theorem 3.16, we may get up to $|E|_\Sigma{}^2$ transitions where we have to store up to $R+1$ states for each transitions where R is the maximal rank appearing in the ranked alphabet Σ. Hence, we compute the transitions in an $\mathcal{O}(R \cdot |E|_\Sigma{}^2)$ space and time complexity.

Overall complexity

Let R be the maximal rank appearing in the ranked alphabet Σ. Then the algorithm sketched above runs altogether with an $\mathcal{O}(R \cdot \mathrm{size}(E)^2)$ space and time complexity.

Example 4.3. Consider the expression E from Example 2.3. Its syntax-tree is shown in Figure 2. We give a linearization $\overline{E}$ of E by labelling the letters with their positions in the syntax-tree:

$$\overline{E} = \Big(\Big(f_{00}\big(g_{000}\big(h_{0000}(a)\big), g_{001}(b)\big)^{*a}\Big) \cdot_b \big(h_{10}(a) + h_{11}(b)\big)\Big).$$

We demonstrate the computation of the partial derivatives, together with their Σ_0-semantics and their FIRST-sets in Figure 3. Here, the identification procedure described above does not conclude into any identification of two D_p. Hence, we get as state set:

$$Q = \{D_\varepsilon, D_{000}, D_{001}, D_{00000}, D_{0000}, D_{0010}, D_{100}, D_{110}\}.$$

p	Label	E_p	Partial derivatives	Σ_0	FIRST
00000	a	$E_{00000} = a$			
0000	h_{0000}	$E_{0000} = h(a)$	$D_{00000} := E_{00000}$	a	$\emptyset$
000	g_{000}	$E_{000} = g(h(a))$	$D_{0000} := E_{0000}$	$\emptyset$	h_{0000}
			$D_{00000} := D_{00000}$	a	$\emptyset$
0010	b	$E_{0010} = b$			
001	g_{001}	$E_{001} = g(b)$	$D_{0010} := E_{0010}$	b	$\emptyset$
00	f_{00}	$E_{00} = f(g(h(a)), g(b))$	$D_{000} := E_{000}$	$\emptyset$	g_{000}
			$D_{001} := E_{001}$	$\emptyset$	g_{001}
			$D_{00000} := E_{00000}$	a	$\emptyset$
			$D_{0000} := E_{0000}$	$\emptyset$	h_{0000}
			$D_{0010} := E_{0010}$	b	$\emptyset$
0	$*a$	$E_0 = \left((f(g(h(a)), g(b)))^* \right)$	$D_{000} := (E_{000} \cdot_a E_0)$	$\emptyset$	g_{000}
			$D_{001} := (E_{001} \cdot_a E_0)$	$\emptyset$	g_{001}
			$D_{00000} := (E_{00000} \cdot_a E_0)$	a	f_{00}
			$D_{0000} := (E_{0000} \cdot_a E_0)$	$\emptyset$	h_{0000}
			$D_{0010} := (E_{0010} \cdot_a E_0)$	b	$\emptyset$
100	a	$E_{100} = a$			
10	h_{10}	$E_{10} = h(a)$	$D_{100} := E_{100}$	a	$\emptyset$
110	b	$E_{110} = b$			
11	h_{11}		$D_{110} := E_{110}$	b	$\emptyset$
1	$+$	$E_1 = (E_{10} + E_{11})$	$D_{100} := E_{100}$	a	$\emptyset$
			$D_{110} := E_{110}$	b	$\emptyset$
ε	$\cdot_b$	$E_\varepsilon = E$	$D_\varepsilon := E_\varepsilon$	a	f_{00}
			$D_{000} := ((E_{000} \cdot_a E_0) \cdot_b E_1)$	$\emptyset$	g_{000}
			$D_{001} := ((E_{001} \cdot_a E_0) \cdot_b E_1)$	$\emptyset$	g_{001}
			$D_{00000} := ((E_{00000} \cdot_a E_0) \cdot_b E_1)$	a	f_{00}
			$D_{0000} := ((E_{0000} \cdot_a E_0) \cdot_b E_1)$	$\emptyset$	h_{0000}
			$D_{0010} := ((E_{0010} \cdot_a E_0) \cdot_b E_1)$	$\emptyset$	h_{10}, h_{11}
			$D_{100} := E_{100}$	a	$\emptyset$
			$D_{110} := E_{110}$	b	$\emptyset$

FIGURE 3. Computing the partial derivatives.

Computing the transitions as shown above, we obtain:

$$D_\varepsilon \xrightarrow{f} (D_{000}, D_{001}), \qquad D_{000} \xrightarrow{g} D_{0000}, \qquad D_{001} \xrightarrow{g} D_{0010},$$
$$D_{0000} \xrightarrow{h} D_{00000}, \qquad D_{0010} \xrightarrow{h} D_{100}, \qquad D_{0010} \xrightarrow{h} D_{110},$$
$$D_{00000} \xrightarrow{f} (D_{000}, D_{001}), \qquad D_\varepsilon \xrightarrow{a} \bot, \qquad D_{00000} \xrightarrow{a} \bot,$$
$$D_{100} \xrightarrow{a} \bot, \qquad D_{110} \xrightarrow{b} \bot.$$

It is not difficult to verify that this automaton is isomorphic to the one from Example 2.7.

REFERENCES

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *Data Structures and Algorithms*. Addison-Wesley, Reading, MA (1983).

[2] V. Antimirov, Partial derivatives of regular expressions and finite automaton constructions. *Theoret. Comput. Sci.* **155** (1996) 291–319.

[3] G. Berry and R. Sethi, From regular expressions to deterministic automata. *Theoret. Comput. Sci.* **48** (1986) 117–126.

[4] J.A. Brzozowski, Derivatives of regular expressions. *J. Assoc. Comput. Mach.* **11** (1964) 481–494.
[5] J.-M. Champarnaud and D. Ziadi, From c-continuations to new quadratic algorithms for automaton synthesis. *Int. J. Algebra Comput.* **11** (2001) 707–735.
[6] J.-M. Champarnaud and D. Ziadi, Canonical derivatives, partial derivatives and finite automaton constructions. *Theoret. Comput. Sci.* **289** (2002) 137–163.
[7] J.-M. Champarnaud, F. Nicart and D. Ziadi, Computing the follow automaton of an expression, in *Proc. of CIAA 2004. Lect. Notes Comput. Sci.* **3317** (2004) 90–101.
[8] H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison and M. Tommasi, Tree automata techniques and applications. Available on: http://www.grappa.univ-lille3.fr/tata, release October, 12th (2007).
[9] F. Gécseg and M. Steinby, Tree languages, in *Handbook of Formal Languages* **3**. Springer (1997) Chap. 1, 1–68.
[10] V.M. Glushkov, The abstract theory of automata. *Russian Mathematical Surveys* **16** (1961) 1–53.
[11] H. Hosoya and B. Pierce, Regular expression pattern matching for XML. *SIGPLAN Not.* **36** (2001) 67–80.
[12] J. Hromkovic, S. Seibert and T. Wilke, Translating regular expressions into small ε-free nondeterministic finite automata. *J. Comput. System Sci.* **62** (2001) 565–588.
[13] L. Ilie and S. Yu, Constructing NFAs by optimal use of positions in regular expressions, in *Proc. of CPM 2002. Lect. Notes Comput. Sci.* **2373** (2002) 279–288.
[14] S.E. Kleene, Representations of events in nerve nets and finite automata, in *Automata Studies*. C.E. Shannon and J. McCarthy, Eds. Princeton University Press (1956) 3–42.
[15] D. Kuske and I. Meinecke, Construction of tree automata from regular expressions, in *Proc. of DLT 2008. Lect. Notes Comput. Sci.* **5257** (2008) 491–503.
[16] S. Lombardy and J. Sakarovitch, Derivatives of rational expressions with multiplicity. *Theoret. Comput. Sci.* **332** (2005) 141–177.
[17] R.F. McNaughton and H. Yamada, Regular expressions and state graphs for automata. *IEEE Trans. Electron. Comput.* **9** (1960) 39–57.
[18] R. Paige and R.E. Tarjan, Three partition refinement algorithms. *SIAM J. Comput.* **16** (1987) 973–989.
[19] J. Sakarovitch, *Éléments de théorie des automates*. Vuibert (2003).
[20] J. Sakarovitch, The language, the expression, and the (small) automaton, in *Implementation and Application of Automata, 10th International Conference, CIAA 2005. Lect. Notes Comput. Sci.* **3845** (2005) 15–30.
[21] T. Suzuki and S. Okui, Hedge pattern partial derivative, in *Proc. of CIAA 2009. Lect. Notes Comput. Sci.* **5642** (2009) 125–134.
[22] J.W. Thatcher and J.B. Wright, Generalized finite automata theory with application to a decision problem of second-order logic. *Math. Syst. Theor.* **2** 57–81 (1968).
[23] P. Van Emde Boas, Machine models and simulations, in *Handbook of Theoretical Computer Science* **A**. Elsevier & The MIT Press (1990) Chap. 1, 1–66.
[24] D. Ziadi, J.-P. Ponty and J.-M. Champarnaud, Passage d'une expression rationnelle à un automate fini non-déterministe. *Bull. Belg. Math. Soc.* **4** (1997) 177–203.

Communicated by C. Choffrut.
Received May 26, 2009. Accepted April 7, 2011.