RAIRO-Theor. Inf. Appl. 55 (2021) 4
https://doi.org/10.1051/ita/2021008

RAIRO - Theoretical Informatics and Applications
www.rairo-ita.org

BOTTOM-UP DERIVATIVES OF TREE EXPRESSIONS

Samira Attou[1,*], Ludovic Mignot[2,3] and Djelloul Ziadi[2,3]

Abstract. In this paper, we extend the notion of (word) derivatives and partial derivatives due to (respectively) Brzozowski and Antimirov to tree derivatives using already known inductive formulae of quotients. We define a new family of extended regular tree expressions (using negation or intersection operators), and we show how to compute a Brzozowski-like inductive tree automaton; the fixed point of this construction, when it exists, is the derivative tree automaton. Such a deterministic tree automaton can be used to solve the membership test efficiently: the whole structure is not necessarily computed, and the derivative computations can be performed in parallel. We also show how to solve the membership test using our (Bottom-Up) partial derivatives, without computing an automaton.

Mathematics Subject Classification. 68Q45.

Received December 17, 2019. Accepted May 26, 2021.

1. Introduction

In 1956, Kleene [8] gave a fundamental theorem in automata theory. He showed that every regular expression E can be converted into a finite state machine that recognizes the same language as E, and *vice versa*. A lot of methods have been proposed to provide the conversion of a given regular expression to a finite word automaton. One of these approaches which appeared in 1964 was Brzozowski's [3] construction; the idea is to use the notion of derivation to compute a deterministic automaton: the derivative of a regular expression E w.r.t. a word w is a regular expression that denotes the set of words w' such that ww' is denoted by E. This construction is not necessarily finite: the derivatives of a given regular expression may form an infinite set. However, considering three equivalence rules (associativity, commutativity and idempotence of the sum), he proved that the set of (so called) similar derivatives is finite.

Antimirov [1], in 1996 introduced the partial derivation which is a similar operation to the one defined by Brzozowski; a partial derivative of a regular expression is no longer a regular expression but a set of regular expressions, that leads to the construction of a non-deterministic automaton, with at most $(n+1)$ states where n is the number of letters of the regular expression. However, this operation is not defined for extended expressions (*i.e.* regular expressions with negation or intersection operators).[1]

Keywords and phrases: Regular tree expressions, derivatives tree automata, partial derivatives, bottom-Up derivatives.

[1] USTHB, Faculty of Mathematics, RECITS Laboratory, BP 32, El Alia, 16111 Bab Ezzouar, Algiers, Algeria.
[2] Groupe de Recherche Rouennais en Informatique Fondamentale, Université de Rouen Normandie, Avenue de l'Université, 76801 Saint-Étienne-du-Rouvray, France.
[3] Associated Member of RECITS Laboratory, CATI Team, USTHB, Algiers, Algeria.

* Corresponding author: sattou@usthb.dz

[1] This was achieved by Caron *et al.* using clausal forms instead of sets [5].

Some of these constructions have been extended to tree automata [12]. Kuske and Meinecke [9] in 2011, introduced an algorithm to convert a regular tree expression into a non-deterministic tree automaton in a Top-Down interpretation. This construction was inspired by Antimirov's construction. In 2017, Champarnaud *et al.* [6] have extended the inductive formulas of quotients to tree languages, following a Bottom-Up interpretation. These notions of derivatives and quotients have practical and theoretical aspects. From a practical point of view, this Bottom-Up interpretation can be related to the notion of contexts of trees, that have been studied in functional programming (*e.g.* zippers [7]). From a theoretical point of view, this study belongs to a large project that aims to study the algorithmic similarities between word automata and tree automata in order to generalize these notions over other algebraic structures [10].

In this paper, we define a new construction of tree automata based on the notion of derivation of an extended tree expression. We also show how to extend the notion of partial derivation in a Bottom-Up way and that the previous construction cannot be applied directly with partial derivatives. Notice that we leave the (technical) study of the finiteness of the set of derivatives and partial derivatives for a future work.

This paper, which is an extended version of [2], is structured as follows: Section 2 defines preliminaries and notations considered throughout this paper. In Section 3, we recall the Bottom-Up quotient formulas for trees and for languages defined in [6]. We explain in Section 4 how we can deal with the Boolean operations. In Section 5, we define the derivative formulas for an extended tree expression. Using the sets of derivatives, in Section 6, we show how to compute a deterministic tree automaton from an extended tree expression that recognizes the same language. We then extend the computation formulae to deal with sets of expressions instead of a single expression and show their validity in Section 7. In Section 8, we present a web application allowing the computation of Bottom-Up derivatives, partial derivatives, and both the derivative tree automaton and a classical non-deterministic inductive construction, where the complement is performed *via* determinization.

2. Preliminaries

Let us first introduce some notations and preliminary definitions. For any pattern matching, we denote by $\_$ the wildcard.

In the following of this paper, we consider a *ranked alphabet* $\Sigma = \bigcup_{k\in\mathbb{N}} \Sigma_k$, *i.e.* a finite graded set of distinct symbols. A symbol f in Σ_k is said to be *k-ary*.

A *tree* t over Σ is inductively defined by $t = \varepsilon_j$ (an empty tree with no symbols) or $t = f(t_1, \ldots, t_n)$, where j is a positive integer, f is a symbol in Σ_n, and $t_1, \ldots, t_n$ are n trees over Σ. Moreover, we assume that for any integer j, the symbol ε_j appears at most once in a tree. We denote by $\mathrm{Ind}_\varepsilon(t)$ the (naturally ordered) set of integers j such that ε_j appears in the tree t. The tree t is *k-ary* if k is the cardinal of $\mathrm{Ind}_\varepsilon(t)$; Precisely, t is *nullary* if $\mathrm{Ind}_\varepsilon(t)$ is empty. Given an integer z, we denote by $\mathrm{Inc}_\varepsilon(z, t)$ the substitution of all symbols ε_x by ε_{x+z} in t. A tree language (a set of trees) L is *k-homogeneous* if it only contains k-ary trees t with the same ε-index (ordered) set, denoted by $\mathrm{Ind}_\varepsilon(L)$ in this case. The language L is *homogeneous* if it is k-homogeneous for some k. We denote by $T(\Sigma)$ the set of the trees over Σ, and $T(\Sigma)_k$ the set of k-ary trees over Σ.

Example 2.1. Let us consider a ranked alphabets $\Sigma = \Sigma_2 \cup \Sigma_1 \cup \Sigma_0$ where $f \in \Sigma_2$, $g \in \Sigma_1$ and $a \in \Sigma_0$. Let $t = f(f(t', \varepsilon_3), \varepsilon_1)$ and $t' = g(a)$ be two trees over Σ. Then, t is 2-ary because $|Ind_\varepsilon(t)| = |\{1, 3\}| = 2$ and t' is 0-ary because $|Ind_\varepsilon(t')| = |\emptyset| = 0$.

Moreover, let us consider the languages $L_1 = \{f(a, a),\ a\}$, $L_2 = \{f(a, \varepsilon_1),\ \varepsilon_1\}$ and $L_3 = \{f(\varepsilon_1, \varepsilon_2)\}$. Then, L_1 contains only 0-ary trees than it is 0-homogeneous, L_2 contains only 1-ary trees then it is 1-homogeneous and last L_3 contains only 2-ary trees then it is 2-homogeneous.

Given a tree t over an alphabet Σ with $\mathrm{Ind}_\varepsilon(t) = \{e_1, \ldots, e_k\}$ and k trees $t_1, \ldots, t_k$ over Σ, we denote by $t \circ (t_1, \ldots, t_k)$ the tree obtained by substituting each ε_{e_i} by t_i in t. Given a k-homogeneous language L over Σ and k languages $(L_1, \ldots, L_k)$ over Σ, we denote by $\circ$ the operation defined by

$$L \circ (L_1, \ldots, L_k) = \{t \circ (t_1, \ldots, t_k) \mid (t, t_1, \ldots, t_k) \in L \times L_1 \times \cdots \times L_k\}. \tag{2.1}$$

Let L be a 1-homogeneous language with $\mathrm{Ind}_\varepsilon(L) = \{j\}$. We denote by L^n the language inductively defined by $L^0 = \{\varepsilon_j\}, L^n = L \circ L^{n-1}$, for any integer $n > 0$, and we set

$$L^{\circledast} = \bigcup_{n\in\mathbb{N}} L^n.$$

Given a tree t, a symbol a in Σ_0 and a 0-homogeneous language L', we denote by $t \cdot_a L'$ the tree language inductively defined by

$$b \cdot_a L' = \begin{cases} L' & \text{if } a = b, \\ \{b\} & \text{otherwise,} \end{cases} \qquad \varepsilon_j \cdot_a L' = \{\varepsilon_j\},$$
$$f(t_1, \ldots, t_n) \cdot_a L' = f(t_1 \cdot_a L', \ldots, t_n \cdot_a L'),$$

with b a tree in Σ_0, f a symbol in Σ_n, $f(L_1, \ldots, L_n) = \{f(t_1, \ldots, t_n) \mid (t_1, \ldots, t_n) \in L_1 \times \cdots \times L_n\}$ and n trees $t_1, \ldots, t_n$ over Σ. Moreover, given a homogeneous language L, we set

$$L \cdot_a L' = \bigcup_{t\in L} t \cdot_a L'.$$

Finally, let us denote by $L^{a,n}$ the language inductively defined by $L^{a,0} = \{a\}$ and $L^{a,n} = L^{a,n-1} \cup L \cdot_a L^{n-1}$ for any integer $n > 0$, and we set

$$L^{*_a} = \bigcup_{n\in\mathbb{N}} L^{a,n}.$$

Example 2.2. Let us consider the 1-homogeneous languages $L = \{f(a, \varepsilon_1)\}$ and $L_1 = \{g(\varepsilon_1), \varepsilon_1\}$ over $\Sigma = \Sigma_2 \cup \Sigma_1 \cup \Sigma_0$ where $f \in \Sigma_2$, $g \in \Sigma_1$ and $a \in \Sigma_0$.

$$\begin{aligned} L \circ (L_1) &= \{f(a, g(\varepsilon_1)),\ f(a, \varepsilon_1)\}, \\ L^{a,1} &= L^{a,0} \cup \{f(a, \varepsilon_1)\} \cdot_a L^{a,0} \\ &= \{a\} \cup \{f(a, \varepsilon_1)\} \cdot_a \{a\} \\ &= \{a,\ f(a, \varepsilon_1)\}, \end{aligned} \qquad \begin{aligned} L^{a,0} &= \{a\}, \\ L^{a,2} &= L^{a,1} \cup f(a, \varepsilon_1) \cdot_a L^{a,1} \\ &= \{a,\ f(a, \varepsilon_1)\} \cup \{f(a, \varepsilon_1),\ f(f(a, \varepsilon_1), \varepsilon_1\} \\ &= \{a,\ f(a, \varepsilon_1),\ f(f(a, \varepsilon_1), \varepsilon_1))\}, \end{aligned}$$

$$L^{*_a} = \{a,\ f(a, \varepsilon_1),\ f(f(a, \varepsilon_1), \varepsilon_1)), \ldots\}.$$

$$\begin{aligned} L^0 &= \{\varepsilon_1\}, \\ L^2 &= L \circ L^1 \\ &= \{f(a, \varepsilon_1)\} \circ \{f(a, \varepsilon_1)\} \\ &= \{f(a, f(a, \varepsilon_1))\}, \end{aligned} \qquad \begin{aligned} L^1 &= L \circ L^0 \\ &= \{f(a, \varepsilon_1)\} \circ \{\varepsilon_1\} \\ &= \{f(a, \varepsilon_1)\}, \end{aligned}$$

$$L^{\circledast} = \{\varepsilon_1,\ f(a, \varepsilon_1),\ f(a, f(a, \varepsilon_1)), \ldots\}.$$

A *tree automaton* over Σ is a 4-tuple $\mathrm{A} = (\Sigma, Q, F, \delta)$ where Q is a set of states, $F \subseteq Q$ is the set of final states, and $\delta \subseteq \bigcup_{k\geq 0}(Q^k \times \Sigma_k \times Q)$ is the set of transitions, which can be seen as the function from $Q^k \times \Sigma_k$

to 2^Q defined by

$$(q_1, \ldots, q_k, f, q) \in \delta \Leftrightarrow q \in \delta(q_1, \ldots, q_k, f).$$

It can be linearly extended as the function from $(2^Q)^k \times \Sigma_k$ to 2^Q defined by

$$\delta(Q_1, \ldots, Q_k, f) = \bigcup_{(q_1,\ldots,q_k)\in Q_1\times\cdots Q_k} \delta(q_1, \ldots, q_k, f). \tag{2.2}$$

Finally, we also consider the function Δ from $T(\Sigma)$ to 2^Q defined by

$$\Delta(f(t_1, \ldots, t_n)) = \delta(\Delta(t_1), \ldots, \Delta(t_n), f). \tag{2.3}$$

Using these definitions, the language $L(A)$ recognized by the tree automaton A is the language $\{t \in T(\Sigma) \mid \Delta(t) \cap F \neq \emptyset\}$.

A tree automaton $A = (\Sigma, Q, F, \delta)$ is *deterministic* if for any symbol f in Σ_m, for any m states $q_1, \ldots, q_m$ in Q, $|\delta(q_1, \ldots, q_m, f)| \leq 1$.

3. Tree language quotients

In this section, we recall the inductive definition of the computation of tree quotients defined in [6].

Let (t, t') be two trees in $T(\Sigma)_k \times T(\Sigma)_{k'}$ such that $=\mathrm{Ind}_\varepsilon(t) \subseteq \mathrm{Ind}_\varepsilon(t')$. Let $R = \mathrm{Ind}_\varepsilon(t)$, $R' = \mathrm{Ind}_\varepsilon(t')$, and $\{(x_z)_{1\leq z\leq k'-k}\} = R' \setminus R$. The *quotient of* t' w.r.t. t is the $(k'-k+1)$-homogeneous tree language $t^{-1}(t')$ that contains all the trees t'' satisfying the two following conditions:

$$t' = t'' \circ (t, (\varepsilon_{x_z})_{1\leq z\leq k'-k}), \quad \mathrm{Ind}_\varepsilon(t'') = \{1, (x_z + 1)_{1\leq z\leq k'-k}\} \tag{3.1}$$

As a direct consequence,

$$\varepsilon_j^{-1}(\varepsilon_l) = \begin{cases} \varepsilon_1 & \text{if } j = l, \\ \emptyset & \text{otherwise.} \end{cases} \tag{3.2}$$

$$t^{-1}(t') = \{\varepsilon_1\} \Leftrightarrow t = t'. \tag{3.3}$$

Definition 3.1. The *Bottom-Up quotient* $t^{-1}(L)$ of a tree language L w.r.t. a tree t is the tree language $\bigcup_{t'\in L} t^{-1}(t')$.

Example 3.2. Let us consider the graded alphabet defined by $\Sigma_2 = \{f\}$, $\Sigma_1 = \{g\}$ and $\Sigma_0 = \{a\}$. Let $t = g(a)$ and $t' = f(f(g(a), \varepsilon_1)), g(a))$ be two trees over $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$. Then

$$t^{-1}(t') = f(f(\varepsilon_1, \varepsilon_2), g(a)),\ f(f(g(a), \varepsilon_2), \varepsilon_1)\}.$$

Notice that for any tree t'' from the set $t^{-1}(t')$, $\quad t'' \circ (g(a), \varepsilon_1) = f(f(g(a), \varepsilon_1)), g(a))$.

As a direct consequence of equation (3.3), the membership of a tree in a tree language can be restated in terms of a quotient:

Proposition 3.3 ([6]). *A tree t is in a language L if and only if ε_1 is in $t^{-1}(L)$.*

Let us now make explicit the inductive computation formulae for this quotient operation. The base cases are the three following ones.

Proposition 3.4 (Prop. 7 of [6]). *Let Σ be a ranked alphabet, k be an integer, and α be in Σ_k:*

$$\alpha^{-1}(\varepsilon_x) = \emptyset, \qquad \alpha^{-1}(\alpha(\varepsilon_1, \dots, \varepsilon_n)) = \{\varepsilon_1\},$$
$$\alpha^{-1}(f(t_1, \dots, t_n)) = \bigcup_{1 \leq j \leq n} f(\{t'_1\}, \dots, \{t'_{j-1}\}, \alpha^{-1}(\{t_j\}), \{t'_{j+1}\}, \dots, \{t'_n\}),$$

where x is an integer in $\mathbb{N}$, f is a symbol in Σ_n, $t_1, \dots, t_n$ are n trees in T_Σ distinct from $(\varepsilon_1, \dots, \varepsilon_n)$ and for all integer $1 \leq z \leq n$, t'_z is the tree $\mathrm{Inc}_\varepsilon(1, t_z)$.

By equation (3.1) and Definition 3.1, quotienting by an indexed ε is reindexing all the indexed ε in the language.

Proposition 3.5 (Prop. 9 of [6]). *Let L be homogeneous with $\mathrm{Ind}_\varepsilon(L) = \{j_1, \dots, j_k\}$ and j be an integer:*

$$\varepsilon_j^{-1}(L) = \begin{cases} L \circ (\varepsilon_{j_1+1}, \dots, \varepsilon_{j_{z-1}+1}, \varepsilon_1, \varepsilon_{j_{z+1}+1}, \dots, \varepsilon_{j_k+1}) & \text{if } j = j_z \in \mathrm{Ind}_\varepsilon(L), \\ \emptyset & \text{otherwise.} \end{cases} \tag{3.4}$$

Example 3.6. Let us consider a tree $t = f(\varepsilon_2, f(a, a))$ with $f \in \Sigma_2$ and $a \in \Sigma_0$. Let us calculate $t^{-1}(t)$. Then

$$\begin{aligned} a^{-1}(t) &= \{f(\varepsilon_3, f(\varepsilon_1, a)),\ f(\varepsilon_3, f(a, \varepsilon_1))\} \\ a^{-1}(a^{-1}(t)) &= \{f(\varepsilon_4, f(\varepsilon_2, \varepsilon_1)),\ f(\varepsilon_4, f(\varepsilon_1, \varepsilon_2)) \circ (\varepsilon_1, \varepsilon_2)\} \\ f(a, a)^{-1}(t) &= \{f(\varepsilon_5, \varepsilon_1) \circ (\varepsilon_1, \varepsilon_2)\} \\ &= \{f(\varepsilon_2, \varepsilon_1)\} \\ f(\varepsilon_2, f(a, a))^{-1}(t) &= \{\varepsilon_1\}. \end{aligned}$$

As a direct consequence of Definition 3.1, the Bottom-Up quotient for the union of languages can be computed as follows:

Lemma 3.7 (Lem. 13 of [6]). *Let t be a tree in $T(\Sigma)$, L_1 and L_2 be two languages over Σ. Then:*

$$t^{-1}(L_1 \cup L_2) = t^{-1}(L_1) \cup t^{-1}(L_2). \tag{3.5}$$

Corollary 3.8 (Cor. 14 of [6]). *Let $t = f(t_1, \dots, t_k)$ be an l-ary tree such that f is in Σ_k and $(t_1, \dots, t_k)$ is a k-tuple of trees in $T(\Sigma)$ different from $(\varepsilon_1, \dots, \varepsilon_k)$. Let L be a n-homogeneous tree language over Σ with $\mathrm{Ind}_\varepsilon(L) = \{x_1, \dots, x_n\}$. Let $\{y_1, \dots, y_{n-l}\} = \mathrm{Ind}_\varepsilon(L) \setminus \mathrm{Ind}_\varepsilon(t)$ and $\forall 1 \leq j \leq k$, $t'_j = \mathrm{Inc}_\varepsilon(k - j, t_j)$. Then:*

$$t^{-1}(L) = (f^{-1}({t'_1}^{-1}(\cdots({t'_k}^{-1}(L))\cdots)) \circ (\varepsilon_1, (\varepsilon_{y_z+1})_{1 \leq z \leq n-l}). \tag{3.6}$$

The Bottom-Up quotient for the b-product of languages can be computed as follows:

Proposition 3.9 (Prop. 17 of [6]). *Let Σ be an alphabet. Let L_1 be a k-homogeneous language, L_2 be a 0-homogeneous language, α be a symbol in Σ and b be a symbol in Σ_0. Then:*

$$\alpha^{-1}(L_1 \cdot_b L_2) = \begin{cases} (b^{-1}(L_1) \cdot_b L_2) \circ_1 b^{-1}(L_2) & \text{if } \alpha = b, \\ \alpha^{-1}(L_1) \cdot_b L_2 \cup (b^{-1}(L_1) \cdot_b L_2) \circ_1 \alpha^{-1}(L_2) & \text{if } \alpha \in \Sigma_0 \setminus \{b\}, \\ \alpha^{-1}(L_1) \cdot_b L_2 & \text{otherwise,} \end{cases}$$

where $\circ_1$ *is the partial composition defined by* $L \circ_1 L' = L \circ (L', (\varepsilon_l)_{j_2 \leq l \leq j_k})$ *with* $\mathrm{Ind}_\varepsilon(L) = \{j_1, \ldots, j_k\}$.

The Bottom-Up quotient for the composition of languages can be computed as follows:

Proposition 3.10 (Prop. 20 of [6]). *Let* Σ *be an alphabet. Let* L *be a* k*-homogeneous language with* $\mathrm{Ind}_\varepsilon(L) = \{j_1, \ldots, j_k\}$, $L_1, \ldots, L_k$ *be* k *tree languages and* α *be in* Σ_n. *Then:*

$$\alpha^{-1}(L \circ (L_1, \ldots, L_k)) = \bigcup_{1 \leq j \leq k} L \circ ((\mathrm{Inc}_\varepsilon(1, L_l))_{1 \leq l \leq j}, \alpha^{-1}(L_j), (\mathrm{Inc}_\varepsilon(1, L_l))_{j+1 \leq l \leq k}) \cup \begin{cases} \alpha((\varepsilon_{j_{p_l}})_{1 \leq l \leq n})^{-1}(L) \circ (\varepsilon_1, (\mathrm{Inc}_\varepsilon(1, L_l))_{1 \leq l \leq k \mid \forall z, l \neq p_z}) \\ \quad \text{if } \forall 1 \leq l \leq n, \exists 1 \leq p_l \leq k, \varepsilon_l \in L_{p_l} \\ \emptyset \quad \text{otherwise.} \end{cases} \tag{3.7}$$

Let us explain the above formula, in order to explain the quotient of the composition of the set of k-ary trees L with k-homogeneous languages $L_1, \ldots, L_k$ w.r.t. a symbol α we first explain $\alpha^{-1}(t \circ (t_1, \ldots, t_k))$.

The composition of a k-ary tree t such that $Ind_\varepsilon(t) = \{x_1, \ldots, x_k\}$, with k trees $t_1, \ldots, t_k$ is the action of grafting these trees to t at the positions where the symbols $\varepsilon_{x_1}, \ldots, \varepsilon_{x_k}$ appear. Thus, the obtained tree t' can be seen as a tree with an upper part containing t and lower parts containing exactly the trees $t_1, \ldots, t_k$. Therefore, if α appears in a lower tree t_j, this tree must be quotiented w.r.t. α and the other parts are ε-incremented. Moreover, if some n trees in $t_1, \ldots, t_k$ are equal to $\varepsilon_1, \ldots, \varepsilon_n$, for example $t_{p_1}, \ldots, t_{p_n}$, and if $t' = \alpha(\varepsilon_{x_{p_1}}, \ldots, \varepsilon_{x_{p_n}})$ appears in t, then t' must be substituted by ε_1 and the other lower trees t_j with $j \neq p_m$, $m \in \{1, \ldots, n\}$ ε-incremented, since the inverse operation produces t. Therefore, by Definition 2.1 we can extend this operation to the case of languages and we find the above formula.

The Bottom-Up quotient for the composition closure of a language can be computed as follows:

Proposition 3.11 (Prop. 22 of [6]). *Let* L *be a* 1*-homogeneous language. Let* α *be a symbol in* $\Sigma_0 \cup \Sigma_1$. *Then:*

$$\alpha^{-1}(L^{\circledast}) = \begin{cases} (L^{\circledast} \circ (\alpha^{-1}(L))) \circ (\varepsilon_1, \mathrm{Inc}_\varepsilon(1, L^{\circledast})) & \text{if } \alpha \in \Sigma_0, \\ (L^{\circledast} \circ (\alpha^{-1}(L))) & \text{otherwise.} \end{cases}$$

The Bottom-Up quotient for the iterated composition of a language can be computed as follows:

Proposition 3.12 (Prop. 24 of [6]). *Let* L *be a* 0*-homogeneous language. Let* α *and* b *be two symbols in* Σ_0. *Then:*

$$\alpha^{-1}(L^{*_b}) = \begin{cases} (b^{-1}(L))^{\circledast} \cdot_b L^{*_b} & \text{if } \alpha = b, \\ ((b^{-1}(L))^{\circledast} \circ (\alpha^{-1}(L))) \cdot_b L^{*_b} & \text{otherwise.} \end{cases}$$

4. Boolean (homogeneous) operations

In the following, we consider that Boolean operations (such as union, intersection, complement, *etc.*) are not necessarily defined for all combinations of languages. Instead, we will consider particular restrictions of these operations, based on the combination of homogeneous languages with the same ε-indices.

As an example, given a k-homogeneous language L, we denote by $\neg L$ the set

$$\{t \in T(\Sigma)_k \mid t \notin L, \mathrm{Ind}_\varepsilon(t) = \mathrm{Ind}_\varepsilon(L)\}. \tag{4.1}$$

By similarly restricting the classical union to pairs of languages with the same ε-indices, one can redefine any Boolean operator as a classical combination of union and complementation (*e.g.* symmetrical difference, set difference, *etc.*). Let us show how to compute the Bottom-Up quotient of a complemented language.

Proposition 4.1. *Let L be an homogeneous language over Σ and $t \in T(\Sigma)$. Then*

$$t^{-1}(\neg L) = \neg(t^{-1}(L)).$$

Proof. Let t'' be a tree in $T(\Sigma)$ such that

$$\mathrm{Ind}_\varepsilon(t'') = \{1, (x_z + 1)_{1 \leq z \leq k'-k}\}.$$

Then

$$\begin{aligned} t'' \in t^{-1}(\neg L) &\Leftrightarrow t'' \circ (t, (\varepsilon_{x_z})_{1 \leq z \leq k-k'}) \in \neg L \\ &\Leftrightarrow t'' \circ (t, (\varepsilon_{x_z})_{1 \leq z \leq k-k'}) \notin L \\ &\Leftrightarrow t'' \notin t^{-1}(L) \\ &\Leftrightarrow t'' \in \neg(t^{-1}(L)). \end{aligned}$$

□

As a direct consequence, following equation (3.5), we get the following result.

Corollary 4.2. *Let $(L_1, \ldots, L_k)$ be k-homogeneous languages with the same ε-indices and let* op *be a Boolean operation. Then for any tree t in $T(\Sigma)$*

$$t^{-1}(\mathrm{op}(L_1, \ldots, L_k)) = \mathrm{op}(t^{-1}(L_1), \ldots, t^{-1}(L_k)).$$

The restriction to homogeneous languages needs a small modification in terms of computation. Indeed, let us consider equation (3.7). It is necessary to determine whether ε_j belongs to a given language: how can we decide whether it belongs to $\neg\emptyset$? There are two alternatives of this barred notation, because the complementation function needs to know the ε-index set of the language it complements. Either we can specify the restriction by parameterizing the operators, or we specify only the occurrences of the empty set symbol, leading to the consideration of expressions instead of languages. This is the approach that we will consider in the following: the languages and the expressions will be subtyped w.r.t. the sets of ε-indices.

5. Extended tree expressions

An *extended tree expression* (tree expression for short) E over Σ is inductively defined by

$$\begin{array}{lll} E = f(E_1, \ldots, E_n), & E = \varepsilon_j, & E = \emptyset_{\mathcal{I}}, \\ E = \mathrm{op}(E_1, \ldots, E_n), & E = E' \circ (E_1, \ldots, E_n), & E = E_1^{\circledast}, \\ E = E_1 \cdot_a E_2, & E = E_1^{*_a}, & \end{array} \tag{5.1}$$

where f is a symbol in Σ_n, $(E', E_1, \ldots, E_n)$ are $(n+1)$ tree expressions over Σ, j is a positive integer, $\mathcal{I}$ is a set of integers, op is an n-ary Boolean operator and a is a symbol in Σ_0. We denote the expression $\emptyset_\emptyset$ by $\emptyset$.

The set of ε-indices $\mathrm{Ind}_\varepsilon(E)$ of a tree expression E, that we use to distinguish tree expressions (*e.g.* distinct occurrences of $\emptyset$), is inductively defined, following equation (5.1), by

$$\begin{aligned}
\mathrm{Ind}_\varepsilon(\varepsilon_j) &= \{j\}, & \mathrm{Ind}_\varepsilon(\emptyset_{\mathcal{I}}) &= \mathcal{I},\\
\mathrm{Ind}_\varepsilon(E_1 \cdot_a E_2) &= \mathrm{Ind}_\varepsilon(E_1) \cup \mathrm{Ind}_\varepsilon(E_2), & \mathrm{Ind}_\varepsilon(E_1^{\circledast}) &= \mathrm{Ind}_\varepsilon(E_1^{*_a}) = \mathrm{Ind}_\varepsilon(E_1).\\
\mathrm{Ind}_\varepsilon(f(E_1,\ldots,E_n)) &= \mathrm{Ind}_\varepsilon(\mathrm{op}(E_1,\ldots,E_n))\\
&= \mathrm{Ind}_\varepsilon(E' \circ (E_1,\ldots,E_n))\\
&= \bigcup_{1\le k\le n} \mathrm{Ind}_\varepsilon(E_k),
\end{aligned}$$

In the following, we restrict the set of tree expressions that we deal with in order to simplify the different computations. More formally, we define the notion of valid tree expression, that rejects (for instance) non-homogeneous tree expressions: A tree expression E is *valid* if it satisfies the predicate $V(E)$ inductively defined, following equation (5.1), by

$$\begin{aligned}
V(\varepsilon_j) = V(\emptyset_{\mathcal{I}}) &= \mathrm{True},\\
V(f(E_1,\ldots,E_n)) &= \Big(\bigwedge_{1\le k\le n} V(E_k)\Big) \wedge \Big(\bigwedge_{1\le k<k'\le n} (\mathrm{Ind}_\varepsilon(E_k) \cap \mathrm{Ind}_\varepsilon(E_{k'}) = \emptyset)\Big),\\
V(\mathrm{op}(E_1,\ldots,E_n)) &= \Big(\bigwedge_{1\le k\le n} V(E_k)\Big) \wedge \Big(\bigwedge_{1\le k<n} (\mathrm{Ind}_\varepsilon(E_k) = \mathrm{Ind}_\varepsilon(E_{k+1}))\Big),\\
V(E' \circ (E_1,\ldots,E_n)) &= V(E') \wedge \Big(\bigwedge_{1\le k\le n} V(E_k)\Big) \wedge (\mathrm{Card}(\mathrm{Ind}_\varepsilon(E')) = n)\\
&\quad \wedge \Big(\bigwedge_{1\le k<k'\le n} (\mathrm{Ind}_\varepsilon(E_k) \cap \mathrm{Ind}_\varepsilon(E_{k'}) = \emptyset)\Big),\\
V(E_1^{\circledast}) &= V(E_1) \wedge (\mathrm{Card}(\mathrm{Ind}_\varepsilon(E_1)) = 1),\\
V(E_1 \cdot_a E_2) &= V(E_1) \wedge V(E_2) \wedge (\mathrm{Ind}_\varepsilon(E_2) = \emptyset),\\
V(E_1^{*_a}) &= V(E_1) \wedge (\mathrm{Ind}_\varepsilon(E_1) = \emptyset).
\end{aligned}$$

The language $L(E)$ *denoted* by a valid tree expression E with an ε-index set $\mathcal{I}$ is inductively defined by

$$\begin{aligned}
L(f(E_1,\ldots,E_n)) &= f(L(E_1),\ldots,L(E_n)), & L(\varepsilon_j) &= \{\varepsilon_j\},\\
L(\mathrm{op}(E_1,\ldots,E_n)) &= \mathrm{op}'(L(E_1),\ldots,L(E_n)), & L(\emptyset_{\mathcal{I}}) &= \emptyset,\\
L(E' \circ (E_1,\ldots,E_n)) &= L(E') \circ (L(E_1),\ldots L(E_n)), & L(E_1^{\circledast}) &= (L(E_1))^{\circledast},\\
L(E_1 \cdot_a E_2) &= L(E_1) \cdot_a L(E_2), & L(E_1^{*_a}) &= (L(E_1))^{*_a},
\end{aligned}$$

where f is a symbol in Σ_n, $(E', E_1, \ldots, E_n)$ are $(n+1)$ tree expressions over Σ, j is a positive integer, op is an n-ary Boolean operator, op' is an n-ary Boolean operation over homogeneous languages with $\mathcal{I}$ as ε-index set (*e.g.* Eq. (4.1)) and a is a symbol in Σ_0. From these definitions, we can define the derivation formulae for valid tree expressions w.r.t. symbols and trees as a syntactical transcription of the quotient formulae.

Definition 5.1. Let E be a valid tree expression and j be an ε-index of E. Then $d_{\varepsilon_j}(E)$ is obtained by incrementing all the ε-indices of E by 1 except ε_j which is replaced by ε_1.

Definition 5.2. Let α be a symbol in Σ_n and F be a valid tree expression over Σ containing $\{1,\dots,n\}$ as ε-indices. The *derivative* of F w.r.t. to α is the expression inductively defined by

$$
\begin{aligned}
d_\alpha(\emptyset_{\mathcal{I}}) &= \emptyset_{\{1\}\cup\{i+1|i>n,i\in\mathcal{I}\}},\\
d_\alpha(\varepsilon_1) &= \emptyset_{\{1\}},\\
d_\alpha(\alpha(\varepsilon_1,\dots,\varepsilon_n)) &= \varepsilon_1,\\
d_\alpha(f(E_1,\dots,E_m)) &= \sum_{1\le j\le n} f(\underline{E_1},\dots,\underline{E_{j-1}},d_\alpha(t_j),\underline{E_{j+1}},\dots,\underline{E_m}),\\
&\quad + \varepsilon_1 \text{ if } \alpha = f \wedge \forall i \le m, \varepsilon_i \in L(E_i),\\
d_\alpha(\mathrm{op}(E_1,\dots,E_k)) &= \mathrm{op}(d_\alpha(E_1),\dots,d_\alpha(E_k)),\\
d_\alpha(E_1 \cdot_b E_2) &= \begin{cases} (d_b(E_1)\cdot_b E_2)\circ_1 d_b(E_2) & \text{if } \alpha = b,\\ d_\alpha(E_1)\cdot_b E_2 + (d_b(E_1)\cdot_b E_2)\circ_1 d_\alpha(E_2) & \text{if } \alpha\in\Sigma_0\setminus\{b\},\\ d_\alpha(E_1)\cdot_b E_2 & \text{otherwise,}\end{cases}\\
d_\alpha(E\circ(E_1,\dots,E_k)) &= \sum_{1\le j\le k} E\circ((\underline{E_l})_{1\le l\le j}, d_\alpha(E_j), (\underline{E_l})_{j+1\le l\le k})\\
&\quad + \begin{cases} d_{\alpha((\varepsilon_{j_{p_l}})_{1\le l\le n})}(E)\circ(\varepsilon_1,(\underline{E_l})_{1\le l\le k|\forall z,l\neq p_z}) \\ \quad \text{if } \forall 1\le l\le n, \exists 1\le p_l\le k, \varepsilon_l\in L(E_{p_l}) \\ \emptyset_{\mathrm{Ind}_\varepsilon(E\circ(E_1,\dots,E_k))\setminus\{1,\dots,n\}} \quad \text{otherwise,}\end{cases}\\
d_\alpha(E^{\circledast}) &= \begin{cases} (E^{\circledast}\circ(d_\alpha(E)))\circ(\varepsilon_1,\mathrm{Inc}_\varepsilon(1,E^{\circledast})) & \text{if } \alpha\in\Sigma_0,\\ (E^{\circledast}\circ(d_\alpha(E))) & \text{otherwise,}\end{cases}\\
d_\alpha(E^{*_b}) &= \begin{cases} (d_b(E))^{\circledast}\cdot_b E^{*_b} & \text{if } \alpha = b,\\ ((d_b(E))^{\circledast}\circ(d_\alpha(E)))\cdot_b E^{*_b} & \text{otherwise,}\end{cases}
\end{aligned}
$$

where $d_{\alpha(\varepsilon_{j_1},\dots,\varepsilon_{j_n})(E)} = d_\alpha(d_{\varepsilon_{j_1+n-1}}(\cdots d_{\varepsilon_{j_{n-1}+1}}(d_{\varepsilon_{j_n}}(E))\cdots))$, where for all integers i the expression $\underline{E_i}$ equals $\mathrm{Inc}_\varepsilon(1,E_i)$ and where $\circ_1$ is the partial composition defined by $E\circ_1 E' = E\circ(E',(\varepsilon_l)_{l\in\{j_2,\dots,j_k\}})$ with $\mathrm{Ind}_\varepsilon(E) = \{j_1,\dots,j_k\}$.

Definition 5.3. Let $t = f(t_1,\dots,t_n)$ be a tree in $T(\Sigma)$ and E a valid tree expression over Σ such that $\mathrm{Ind}_\varepsilon(t)\subseteq \mathrm{Ind}_\varepsilon(E)$. The *derivative* of E w.r.t. t is the tree expression defined by

$$d_t(E) = (d_f(d_{t'_1}(\cdots(d_{t'_k}(E))\cdots))\circ(\varepsilon_1,(\varepsilon_{y_z+1})_{1\le z\le n-l})),$$

where $\{y_1,\dots,y_{n-l}\} = \mathrm{Ind}_\varepsilon(E)\setminus\mathrm{Ind}_\varepsilon(t)$ and $\forall 1\le j\le k$, $t'_j = \mathrm{Inc}_\varepsilon(k-j,t_j)$.

Notice that the base cases include the one of the derivation of the empty set. In this case, the only modification that occurs is the index simulating the ε-index set of the denoted language. This is necessary in order to validate equation (3.1). As an example, consider the expression $E = \neg\emptyset_\emptyset$. When deriving E w.r.t. a nullary tree a, one must obtain an expression denoting all the trees that belong to $T(\Sigma)$ in which one a was removed, that is the set of all the trees with only ε_1 as an ε-index. Applying the previously defined formulae:

$$d_a(E) = \neg\emptyset_{\{1\}}.$$

When deriving one more time w.r.t. a, one must obtain an expression denoting all the trees with only ε_1 and ε_2 as ε-indices (obtained from a tree in $T(\Sigma)$ by removing two occurrences of a). Applying the previously defined

formulae:

$$d_a(d_a(E)) = \neg\emptyset_{\{1,2\}}.$$

Finally, when deriving by a binary symbol f, one must obtain an expression denoting all the trees that belong to $T(\Sigma)$ in which one occurrence of $f(a,a)$ was removed, that is the set of all the trees with only ε_1 as ε-index. Applying the previously defined formulae:

$$d_f(d_a(d_a(E))) = \neg\emptyset_{\{1\}}.$$

As a direct consequence of the inductive formulae of Section 3 and of Corollary 4.2, we get the following theorem.

Theorem 5.4. *The derivative of a valid tree expression E w.r.t. to a tree t denotes $t^{-1}(L(E))$.*

Proof. Let us proceed in three steps.

1. Following equation (3.4) and Definition 5.1, it holds that
$$L(d_{\varepsilon_j}(E)) = \varepsilon_j^{-1}(L(E)).$$
2. Notice that the derivation formulae of Definition 5.2 are syntactical equivalents of the quotient formulae of Section 3 and of Corollary 4.2 and therefore it can be proved by induction over the structure of E that
$$L(d_\alpha(E)) = \alpha^{-1}(L(E)).$$
This reasoning is valid except for the case with indexed occurrence of the empty set and with the case of the substitution product. As discussed before, the occurrences of the empty set are "typed" w.r.t. the ε-indices set of the language they denote. Therefore, these indices should be modified using equation (3.1). In the product case, the derivation of an expression by the tree $\alpha(\varepsilon_{j_1},\dots,\varepsilon_{j_n})$ has to be considered. However, by considering this particular case in the induction, one can check that
$$\begin{aligned} L(d_{\alpha(\varepsilon_{j_1},\dots,\varepsilon_{j_n})}(E)) &= L(d_\alpha(d_{\varepsilon_{j_1+n-1}}(\cdots d_{\varepsilon_{j_{n-1}+1}}(d_{\varepsilon_{j_n}}(E))\cdots))) \\ &= \alpha^{-1}(L(d_{\varepsilon_{j_1+n-1}}(\cdots d_{\varepsilon_{j_{n-1}+1}}(d_{\varepsilon_{j_n}}(E))\cdots))), \end{aligned}$$
this last equality obtained by applying the induction step. From item 1, it holds that
$$\alpha^{-1}(d_{\varepsilon_{j_1+n-1}}(\cdots d_{\varepsilon_{j_{n-1}+1}}(d_{\varepsilon_{j_n}}(E))\cdots)) = \alpha^{-1}({\varepsilon_{j_1+n-1}}^{-1}(\cdots {\varepsilon_{j_{n-1}+1}}^{-1}(L(E))\cdots))$$
that equals $\alpha(\varepsilon_{j_1},\dots,\varepsilon_{j_n})^{-1}(L(E))$ from equation (3.6). We can conclude following equation (3.7).
3. Finally, according to equation (3.6) and item 2, Definition 5.3 implies that
$$L(d_t(E)) = t^{-1}(L(E)).$$

□

Example 5.5. Let us consider the graded alphabet defined by $\Sigma_2 = \{f\}$, $\Sigma_1 = \{g\}$ and $\Sigma_0 = \{a,b,c\}$ and let E be the extended tree expression defined by

$$E = E_1 \cdot_a E_2,$$

with $E_1 = \neg(g(a)^{*_a})$ and $E_2 = f(f(a,a),a)$. Let us show how to calculate the derivative of E w.r.t. $t = f(f(a,a),a)$. First, let us compute the derivative of E_2 w.r.t. t:

$$\begin{aligned}
d_a(E_2) &= f(f(\varepsilon_1, a) + f(a, \varepsilon_1), a) + f(f(a,a), \varepsilon_1),\\
d_a(d_a(E_2)) &= f(f(\varepsilon_2, \varepsilon_1) + f(\varepsilon_1, \varepsilon_2), a) + f(f(\varepsilon_2, a) + f(a, \varepsilon_2), \varepsilon_1)\\
&\quad + f(f(\varepsilon_1, a) + f(a, \varepsilon_1), \varepsilon_2),\\
d_a(d_a(d_a(E_2))) &= f(f(\varepsilon_3, \varepsilon_2) + f(\varepsilon_2, \varepsilon_3), \varepsilon_1) + f(f(\varepsilon_3, \varepsilon_1) + f(\varepsilon_1, \varepsilon_3), \varepsilon_2)\\
&\quad + f(f(\varepsilon_2, \varepsilon_1) + f(\varepsilon_1, \varepsilon_2), \varepsilon_3),\\
d_{f(a,a)}(d_a(E_2)) &= d_{f(\varepsilon_1,\varepsilon_2)}(d_a(d_a(d_a(E_2)))) \circ (\varepsilon_1, \varepsilon_2)\\
&= (\emptyset_{\{1,4\}} + \emptyset_{\{1,4\}} + f(\emptyset_1 + \varepsilon_1, \varepsilon_4)) \circ (\varepsilon_1, \varepsilon_2)\\
&= f(\varepsilon_1, \varepsilon_4) \circ (\varepsilon_1, \varepsilon_2) = f(\varepsilon_1, \varepsilon_2),\\
d_t(E_2) &= d_{f(\varepsilon_1,\varepsilon_2)}(d_{f(a,a)}(d_a(E_2))) = \varepsilon_1.
\end{aligned}$$

Then, in order to reduce the size of the computed tree expressions, let us set

$$E' = \neg(g(\varepsilon_1)^{\circledast}) \cdot_a E_2, \qquad\qquad E'' = \neg(\emptyset_{\{1,2\}}) \cdot_a E_2.$$

Then:

$$\begin{aligned}
d_a(E) &= E' \circ d_a(E_2),\\
d_{f(a,a)}(d_a(E)) &= E'' \circ (f(\varepsilon_1, a), \varepsilon_4) \circ (\varepsilon_1, \mathrm{Inc}_\varepsilon(1, d_a(E_2))) + E' \circ d_{f(a,a)}(d_a(E_2)),\\
d_t(E) &= E'.
\end{aligned}$$

6. Tree automaton construction

In this section, we explain how we can compute a tree automaton from a valid tree expression E with $\mathrm{Ind}_\varepsilon(E) = \emptyset$ from an iterated process using the previously defined derivation.

Given a tree expression E over an alphabet Σ, we first compute the set

$$D_0(E) = \{d_a(E) \mid a \in \Sigma_0\}.$$

From this set, we compute the tree automaton $A_0 = (\Sigma, D_0(E), F_0, \delta_0)$ where

$$F_0 = \{E' \in D_0(E) \mid \varepsilon_1 \in L(E')\}, \qquad\qquad \delta_0 = \{(a, d_a(E)) \mid a \in \Sigma_0\}.$$

From this step, one can choose a total function tree_0 associating any tree expression E' in $D_0(E)$ with a tree t such that

$$\mathrm{tree}_0(E') = t \Rightarrow d_t(E) = E',$$

by choosing for any tree expression E' in $D_0(E)$ a symbol $a \in \Sigma_0$ such that $(a, E') \in \delta_0$. From this induction basis, let us consider the transition set δ_n inductively defined by

$$\begin{aligned}
\delta_n = \{((E'_1, \ldots, E'_m), f, d_t(E)) \mid & t = f(\mathrm{tree}_{n-1}(E'_1), \ldots, \mathrm{tree}_{n-1}(E'_m)),\\
& f \in \Sigma_m,\\
& E'_1, \ldots, E'_m \in D_{n-1}(E)\}.
\end{aligned} \tag{6.1}$$

Let us consider the set $D_n(E) = D_{n-1}(E) \cup \pi_3(\delta_n)$, where π_3 is the classical projection defined by $\pi_3(X) = \{z \mid (\_, \_, z) \in X\}$. Obviously, one can once again choose a total function tree_n associating any tree expression E' in $D_n(E)$ with a tree t such that

$$\text{tree}_n(E') = t \Rightarrow d_t(E) = E',$$

by choosing a transition $((E'_1, \dots, E'_m), f, E')$ in δ_n for any tree expression E' in $D_n(E) \setminus (D_{n-1}(E))$ and defining t as $f(\text{tree}_{n-1}(E'_1), \dots, \text{tree}_{n-1}(E'_k))$. Finally, by considering the set

$$F_n = \{E' \in D_n(E) \mid \varepsilon_1 \in L(E')\}, \tag{6.2}$$

we can define the tree automaton $A_n = (\Sigma, D_n(E), F_n, \delta_n)$.

Let $A(E)$ be the fixed point, if it exists, of this process (up to the choice of the tree_* functions), called the *Bottom-Up derivative tree automaton* of E.

First, notice that the construction leads to a deterministic tree automaton. Let us then state that the validity of the construction does not depend on the choice of the tree_n functions. By a direct induction over the structure of t, considering the definition of δ in equation (7.1), we get the following proposition.

Proposition 6.1. *Let E be a valid tree expression over an alphabet Σ, t be a nullary tree in $T(\Sigma)$ and $A(E) = (\Sigma, Q, F, \delta)$ be a Bottom-Up derivative tree automaton of E. Let $\Delta(t) = \{E'\}$. Then $L(E') = t^{-1}(L(E))$.*

In order to prove this result, let us first show how derivatives behave w.r.t. permutations.

Lemma 6.2. *Let $t_1, \dots, t_k$ be k nullary trees. Let L be a nullary language. Let π be a permutation over $\{1, \dots, k\}$. Then*

$$t_1^{-1}(t_2^{-1}(\cdots {t_k}^{-1}(L)\cdots)) = t_{\pi(1)}^{-1}(t_{\pi(2)}^{-1}(\cdots {t_{\pi(k)}}^{-1}(L)\cdots)) \circ (\varepsilon_{\pi(1)}, \dots, \varepsilon_{\pi(k)}).$$

Proof. Let t be a tree. Then, considering equation (3.1),

$$\begin{aligned}
& t \in t_1^{-1}(t_2^{-1}(\cdots {t_k}^{-1}(L)\cdots)) \\
\Leftrightarrow\ & \exists t' \in L, t' = t \circ (t_1, \dots, t_k) \\
\Leftrightarrow\ & \exists t' \in L, t' = t \circ (\varepsilon_{\pi^{-1}(1)}, \dots, \varepsilon_{\pi^{-1}(k)}) \circ (t_{\pi(1)}, \dots, t_{\pi(k)}) \\
\Leftrightarrow\ & t \circ (\varepsilon_{\pi^{-1}(1)}, \dots, \varepsilon_{\pi^{-1}(k)}) \in t_{\pi(1)}^{-1}(t_{\pi(2)}^{-1}(\cdots {t_{\pi(k)}}^{-1}(L)\cdots)) \\
\Leftrightarrow\ & t \in t_{\pi(1)}^{-1}(t_{\pi(2)}^{-1}(\cdots {t_{\pi(k)}}^{-1}(L)\cdots)) \circ (\varepsilon_{\pi(1)}, \dots, \varepsilon_{\pi(k)})
\end{aligned}$$

□

Let us now show that if two trees act similarly *via* the quotient operation, then their action can be interchanged.

Lemma 6.3. *Let $t_1, \dots, t_k$ be k nullary trees. Let L be a nullary language. Let $1 \leq j \leq k$ be an integer and let t'_j be a nullary tree such that ${t_j}^{-1}(L) = {t'_j}^{-1}(L)$. Then*

$$t_1^{-1}(\cdots t_j^{-1}(\cdots {t_k}^{-1}(L)\cdots)\cdots) = t_1^{-1}(\cdots {t'_j}^{-1}(\cdots {t_k}^{-1}(L)\cdots)\cdots).$$

Proof. Let us consider the permutation π' that only permutes j with k and acts like the identity for the other integers. From Lemma 6.2, one can check the following equivalences:

$$
\begin{aligned}
&t_1^{-1}(\cdots t_j^{-1}(\cdots {t_k}^{-1}(L)\cdots)\cdots)\\
&= {t_{\pi'(1)}}^{-1}(\cdots {t_{\pi'(j)}}^{-1}(\cdots {t_{\pi'(k)}}^{-1}(L)\cdots)\cdots)\circ(\varepsilon_{\pi'(1)},\ldots,\varepsilon_{\pi'(j)},\ldots,\varepsilon_{\pi'(k)})\\
&= {t_1}^{-1}(\cdots {t_k}^{-1}(\cdots {t_j}^{-1}(L)\cdots)\cdots)\circ(\varepsilon_1,\ldots,\varepsilon_{j-1},\varepsilon_k,\varepsilon_{j+1},\ldots,\varepsilon_{k-1},\varepsilon_j)\\
&= {t_1}^{-1}(\cdots {t_k}^{-1}(\cdots {t'_j}^{-1}(L)\cdots)\cdots)\circ(\varepsilon_1,\ldots,\varepsilon_{j-1},\varepsilon_k,\varepsilon_{j+1},\ldots,\varepsilon_{k-1},\varepsilon_j)\\
&= {t_{\pi'(1)}}^{-1}(\cdots {t_{\pi'(j)}}^{-1}(\cdots {t'_{\pi'(k)}}^{-1}(L)\cdots)\cdots)\circ(\varepsilon_{\pi'(1)},\ldots,\varepsilon_{\pi'(j)},\ldots,\varepsilon_{\pi'(k)})\\
&= t_1^{-1}(\cdots {t'_j}^{-1}(\cdots {t_k}^{-1}(L)\cdots)\cdots)
\end{aligned}
$$

□

The repeated application of this lemma trivially leads to the following corollary.

Corollary 6.4. *Let $t_1,\ldots,t_k$ be k nullary trees. Let L be a nullary language. Let $t'_1,\ldots,t'_k$ be k nullary trees such that for any integer $1\leq j\leq k$, ${t_j}^{-1}(L) = {t'_j}^{-1}(L)$. Then*

$$t_1^{-1}(\cdots {t_k}^{-1}(L)\cdots) = {t'_1}^{-1}(\cdots {t'_k}^{-1}(L)\cdots)$$

Finally, we can prove our proposition.

Proof. (of Prop. 6.1) It is sufficient, by definition of the Bottom-Up derivative tree automaton, to prove by recurrence over an integer n that for any tree t of height at most $n+1$, it holds that for $\Delta_n(t)=\{E'\}$,

$$L(E') = t^{-1}(L(E)).$$

1. By definition of δ_0, the equality holds for any tree of height 1.
2. Let us consider that the hypothesis holds at the rank $n-1$. Let $f(t_1,\ldots,t_k)$ be a tree of height $(n+1)$. Obviously, for an integer $1\leq j\leq k$, the height of the tree t_j is smaller than n and therefore, by application of the induction hypothesis, assuming that $\Delta_{n-1}(t_j)=\{E_j\}$,

   $$L(E_j) = {t_j}^{-1}(L(E)).$$

   Moreover, by definition, δ_n contains the transition $((E_1,\ldots,E_k),f,d_t(E))$ where $t=f(\mathrm{tree}_{n-1}(E_1),\ldots,\mathrm{tree}_{n-1}(E_k))$.

   Let us set, for any integer $1\leq j\leq k$, $\mathrm{tree}_{n-1}(E_j)=t'_j$, and therefore $t=f(t'_1,\ldots,t'_k)$.

   It is easy to check, from the construction of the function tree_{n-1} and by a trivial recurrence, that for any state S in A_{n-1},

   $$\Delta_{n-1}(\mathrm{tree}_{n-1}(S)) = \{S\}.$$

   Hence, for any integer $1\leq j\leq k$,

   $$\Delta_{n-1}(t'_j) = \{E_j\}$$

   and by induction hypothesis, since the height of t'_j is at most n, it holds

   $$L(d_{t'_j}(E)) = L(E_j).$$

Consequently,

$$t_j{}^{-1}(L(E)) = t'_j{}^{-1}(L(E)).$$

Notice that from Definition 5.3,

$$d_t(E) = d_f(d_{t'_1}(\cdots d_{t'_k}(E)\cdots)).$$

Hence, from Corollary 6.4 and Theorem 5.4,

$$\begin{aligned} L(d_t(E)) &= L(d_f(d_{t'_1}(\cdots d_{t'_k}(E)\cdots))) \\ &= f^{-1}(t'_1{}^{-1}(\cdots t'_k{}^{-1}(L(E))\cdots)) \\ &= f^{-1}(t_1{}^{-1}(\cdots t_k{}^{-1}(L(E))\cdots)) \\ &= L(d_{f(t_1,\ldots,t_k)}(E)) \end{aligned}$$

□

As a direct consequence of Proposition 6.1, Proposition 3.3 and equation (7.2), the following theorem holds.

Theorem 6.5. *A Bottom-Up derivative tree automaton of a tree expression E is deterministic and recognizes $L(E)$.*

Proof. Let $A = (\Sigma, \_, F, \delta)$ be the Bottom-Up derivative tree automaton of E. By construction, A is deterministic. Let t be a tree over Σ. Then:

$$\begin{aligned} t \in L(E) &\Leftrightarrow \varepsilon_1 \in t^{-1}(L(E)) \\ &\Leftrightarrow \varepsilon_1 \in L(E') \text{ with } \Delta(t) = \{E'\} \\ &\Leftrightarrow E' \in F \text{ with } \Delta(t) = \{E'\} \\ &\Leftrightarrow t \in L(A). \end{aligned}$$

□

Notice that when $\Sigma_n = \emptyset$ for $n \geq 2$, the three ACI rules (associativity, commutativity and idempotence of the sum) are sufficient to obtain a finite tree automaton, isomorphic to the classical Brzozowski automaton (for words). More generally, one can wonder if these three rules are sufficient in order to obtain a finite set of (similar) derivatives. This (technical) study is the next step of our study.

Example 6.6. Let us consider the tree expressions defined in Example 5.5, *i.e.*

$$E = E_1 \cdot_a E_2, \qquad E_1 = \neg(g(a)^{*_a}), \qquad E_2 = f(f(a,a),a), \qquad E' = \neg(g(\varepsilon_1)^{\circledast}) \cdot_a E_2.$$

Let us show how to compute a derivative tree automaton of E. First, let us compute $A_0 = (\Sigma, D_0(E), F_0, \delta_0)$: by definition, $D_0(E) = \{d_a(E), d_b(E), d_c(E)\}$. Hence,

$$\begin{gathered} d_a(E) = E' \circ d_a(E_2), \quad d_b(E) = d_c(E) = \neg(\emptyset_{\{1\}}) \cdot_a E_2, \\ D_0(E) = \{d_a(E), d_b(E)\}, \quad F_0 = \{d_b(E)\}, \quad \delta_0 = \{(a, d_a(E)), (b, d_b(E)), (c, d_b(E))\}. \end{gathered}$$

The tree automaton A_0 is represented in Figure 1, where the state $\emptyset_1$, a sink-state, and its transitions are omitted.

FIGURE 1. The Tree Automaton A_0.

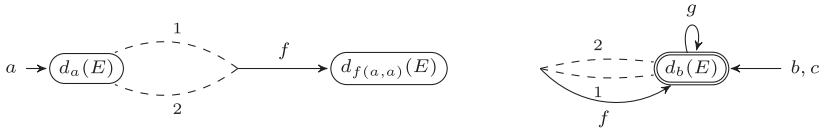

FIGURE 2. The Tree Automaton A_1.

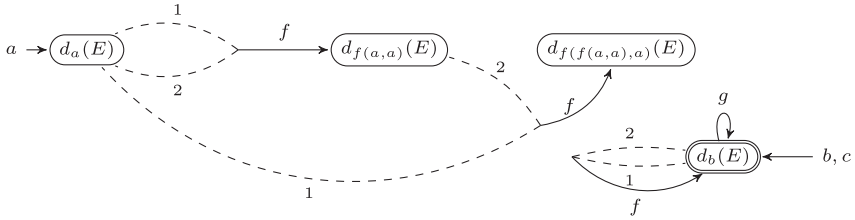

FIGURE 3. The Tree Automaton A_2.

We choose a function to define tree_0: $\text{tree}_0(d_a(E)) = a, \text{tree}_0(d_b(E)) = b$. Let us now show how to compute $A_1 = (\Sigma, D_1(E), F_1, \delta_1)$. According to equation (7.1), it is sufficient to compute the derivatives of E w.r.t. the trees in the set $\{f(a,a), f(a,b), f(b,a), f(b,b), g(a), g(b)\}$:

$$d_{f(a,a)}(E) = E' \circ f(\varepsilon_1, a), \quad d_{f(b,b)}(E) = d_{g(b)}(E) = d_b(E),$$
$$d_{f(a,b)}(E) = d_{f(b,a)}(E) = d_{g(a)}(E) = \emptyset_{\{1\}}.$$

There is a new non-final state, $E' \circ f(\varepsilon_1, a)$, which is associated with $f(a,a)$ by the function tree_1, and three new transitions:

$$\delta_1 = \delta_0 \cup \{(d_a(E), d_a(E), f, d_{f(a,a)}(E)), (d_b(E), d_b(E), f, d_b(E)), (d_b(E), g, d_b(E))\}.$$

The tree automaton A_1 is represented in Figure 2.

The tree automaton A_2 can be computed through the derivatives w.r.t. the trees in the set

$$\{f(f(a,a), f(a,a)), f(f(a,a),a), f(f(a,a),b), f(a, f(a,a)), f(b, f(a,a)), g(f(a,a))\} :$$

$d_{f(f(a,a),b)}(E) = d_{f(a,f(a,a))}(E) = d_{f(b,f(a,a))}(E) = d_{f(b,f(a,a))}(E) = \emptyset_{\{1\}}$, $d_{f(f(a,a),a)}(E) = E'$. There is a new non-final state, E', which is associated with $t = f(f(a,a),a)$ by the function tree_2, and one new transition: $\delta_2 = \delta_1 \cup \{(d_{f(a,a)}(E), d_a(E), f, d_t(E))\}$. The tree automaton A_2 is represented in Figure 3.

The tree automaton A_3 is obtained by computing the derivatives w.r.t. the trees in the set $\{f(t,t), f(t,a), f(t,b), \ldots\}$. There are only four computations that do not return a derivative equal to $\emptyset_{\{1\}}$: $d_{f(t,t)}(E) = d_{f(t,b)}(E) = d_{f(b,t)}(E) = d_b(E)$ and $d_{g(t)}(E) = d_t(E)$. There are four new transitions:

$$\delta_3 = \delta_2 \cup \{(d_t(E), d_t(E), f, d_b(E)), (d_t(E), d_b(E), f, d_b(E)), (d_b(E), d_t(E), f, d_b(E)),$$
$$(d_t(E), g, d_t(E))\},$$

but these computations does not introduce new states. Therefore the computation halts and the derivative tree automaton of E is A_3, represented in Figure 4.

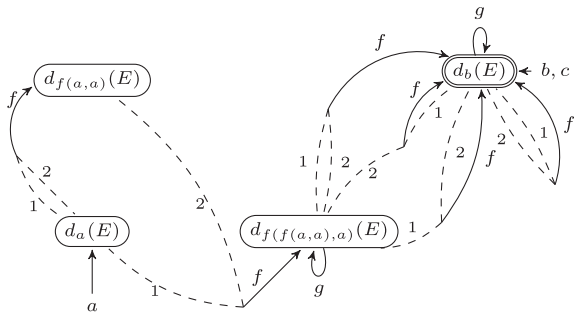

FIGURE 4. The Bottom-Up derivative Tree Automaton of E.

7. Partial derivatives and derived terms

The partial derivation, due to Antimirov [1], is an operation based on the same operation as the derivation. Indeed, they both implement the computation of the quotient at the expression (and then syntactical) level. The main difference is that instead of computing an expression from an expression, the partial derivation produces an expression set from an expression. Computing a sum from this set produces an expression equivalent to the one obtained *via* derivation. It was already extended to tree expressions by Kuske and Meinecke [9] in order to compute a Top-Down automaton. Therefore, let us then show how to extend it in a Bottom-Up way.

As in the case of (Bottom-Up) derivation, let us first explicit the derivation w.r.t. an empty tree. Let E be a valid tree expression and j be an ε-index of E. Then $\partial_{\varepsilon_j}(E)$ is obtained, as in the case of derivation, by incrementing all the ε-indices of E by 1 except ε_j which is replaced by ε_1 *i.e.*

$$\partial_{\varepsilon_j}(E) = \{d_{\varepsilon_j}(E)\}.$$

Its application to an expression set can be easily defined as follows:

$$\partial_{\varepsilon_j}(\mathcal{E}) = \bigcup_{E\in\mathcal{E}} \partial_{\varepsilon_j}(E).$$

The computation of the derived terms, that are the expressions in the set obtained by the partial derivation, can be defined w.r.t. the symbols as follows. Notice that we only deal with the sum as Boolean operator. Classically, the partial derivation is defined for (so-called) simple expressions; however, it can be easily extended to all Boolean operators [4].

Definition 7.1. Let α be a symbol in Σ_n and F be a valid tree expression over Σ containing $\{1,\dots,n\}$ as ε-indices. The *partial derivative* of F w.r.t. to α is the expression inductively defined by

$$\begin{aligned}
\partial_\alpha(\emptyset_{\mathcal{I}}) &= \emptyset,\\
\partial_\alpha(\varepsilon_1) &= \emptyset,\\
\partial_\alpha(\alpha(\varepsilon_1,\dots,\varepsilon_n)) &= \{\varepsilon_1\},\\
\partial_\alpha(f(E_1,\dots,E_m)) &= \bigcup_{1\le j\le m} \{f(\underline{E_1},\dots,\underline{E_{j-1}},E'_j,\underline{E_{j+1}},\dots,\underline{E_m}) \mid E'_j \in \partial_\alpha(E_j)\},\\
&\qquad \cup \{\varepsilon_1 \mid \alpha = f \wedge \forall i \le m, \varepsilon_i \in L(E_i)\}\\
\partial_\alpha(E_1 + E_2) &= \partial_\alpha(E_1) \cup \partial_\alpha(E_2),
\end{aligned}$$

$$\partial_\alpha(E_1 \cdot_b E_2) = \begin{cases} (\partial_b(E_1) \cdot_b E_2) \circ_1 \partial_b(E_2) & \text{if } \alpha = b, \\ \partial_\alpha(E_1) \cdot_b E_2 \cup (\partial_b(E_1) \cdot_b E_2) \circ_1 \partial_\alpha(E_2) & \text{if } \alpha \in \Sigma_0 \setminus \{b\}, \\ \partial_\alpha(E_1) \cdot_b E_2 & \text{otherwise,} \end{cases}$$

$$\begin{aligned} \partial_\alpha(E \circ (E_1, \ldots, E_k)) = & \bigcup_{1 \le j \le k} \{E \circ ((\underline{E_l})_{1 \le l \le j}, E'_j, (\underline{E_l})_{j+1 \le l \le k}) \\ & \mid E'_j \in \partial_\alpha(E_j)\} \\ & \cup \begin{cases} \partial_{\alpha((\varepsilon_{j_{p_l}})_{1 \le l \le n})}(E) \circ (\varepsilon_1, ((\underline{E_l})_{1 \le l \le k \mid \forall z, l \ne p_z}), \\ \qquad \text{if } \forall 1 \le l \le n, \exists 1 \le p_l \le k, \varepsilon_l \in L(E_{p_l}), \\ \emptyset \text{ otherwise,} \end{cases} \end{aligned}$$

$$\partial_\alpha(E^{\circledast}) = \begin{cases} (E^{\circledast} \circ (\partial_\alpha(E))) \circ (\varepsilon_1, \mathrm{Inc}_\varepsilon(1, E^{\circledast})) & \text{if } \alpha \in \Sigma_0, \\ (E^{\circledast} \circ (\partial_\alpha(E))) & \text{otherwise,} \end{cases}$$

$$\partial_\alpha(E^{*_b}) = \begin{cases} (\partial_b(E))^{\circledast} \cdot_b E^{*_b} & \text{if } \alpha = b, \\ ((\partial_b(E))^{\circledast} \circ_1 (\partial_\alpha(E))) \cdot_b E^{*_b} & \text{otherwise,} \end{cases}$$

where

- $\partial_{\alpha(\varepsilon_{j_1}, \ldots, \varepsilon_{j_n})(E)} = \partial_\alpha(\partial_{\varepsilon_{j_1+n-1}}(\cdots \partial_{\varepsilon_{j_{n-1}+1}}(\partial_{\varepsilon_{j_n}}(E)) \cdots))$,
- for all integers i the expression $\underline{E_i}$ equals $\mathrm{Inc}_\varepsilon(1, E_i)$,
- $\mathcal{E} \circ_1 \mathcal{E}' = \{E \circ_1 E' \mid (E, E') \in \mathcal{E} \times \mathcal{E}'\}$,
- $\mathcal{E} \circ (E_1, \ldots, E_m) = \{E \circ (E_1, \ldots, E_m) \mid E \in \mathcal{E}\}$,
- $E \circ_1 \mathcal{E} = \{E \circ_1 E' \mid E' \in \mathcal{E}\}$,
- $\mathcal{E} \cdot_b E = \{E' \cdot_b E \mid E' \in \mathcal{E}\}$,
- $\mathcal{E}^{\circledast} = \{(\sum_{E \in \mathcal{E}} E)^{\circledast}\}$.

The extension of the partial derivation to the operation of negation can be achieved in a similar way to the composition closure: $\partial(\neg E) = \{\neg(\sum_{E' \in \partial(E)} E')\}$.

Another method is to define another partial derivation returning sets of sets of expressions, interpreted as clausal disjunctive forms (an union of intersection of languages) [4, 5].

To well define the operation $\circledast$, an order over the expressions has to be considered. This order is important in order to distinguish between semantic and syntax. As an example, let us consider the set $\{a, b\}^{\circledast}$; considering $a < b$ or $b < a$ leads to two distinct (but equivalent) expressions, $(a + b)^{\circledast}$ and $(b + a)^{\circledast}$. Moreover, we set $\emptyset^{\circledast} = \{\varepsilon_1\}$. Furthermore, the partial derivative of an expression set $\mathcal{E}$ is defined by

$$\partial_\alpha(\mathcal{E}) = \bigcup_{E \in \mathcal{E}} \partial_\alpha(E).$$

Finally, the partial derivative w.r.t. a tree can be defined as follows.

Definition 7.2. Let $t = f(t_1, \ldots, t_n)$ be a tree in $T(\Sigma)$ and E a valid tree expression over Σ such that $\mathrm{Ind}_\varepsilon(t) \subseteq \mathrm{Ind}_\varepsilon(E)$. The *partial derivative* of E w.r.t. t is the set of tree expressions defined by

$$\partial_t(E) = (\partial_f(\partial_{t'_1}(\cdots(\partial_{t'_k}(E))\cdots)) \circ (\varepsilon_1, (\varepsilon_{y_z+1})_{1 \le z \le n-l})),$$

where $\{y_1, \ldots, y_{n-l}\} = \mathrm{Ind}_\varepsilon(E) \setminus \mathrm{Ind}_\varepsilon(t)$ and $\forall 1 \le j \le k$, $t'_j = \mathrm{Inc}_\varepsilon(k - j, t_j)$.

As a direct consequence of the inductive formulae of Section 3, we get the following theorem.

Theorem 7.3. *Let t be a tree in $T(\Sigma)$ and E a valid tree expression over Σ. Then:*

$$\bigcup_{E' \in \partial_t(E)} L(E') = t^{-1}(L(E)).$$

Proof. The proof is similar to the one of Theorem 5.4, by first considering that union distributes over composition (as a direct consequence of Eq. (2.1))

$$\begin{aligned} L \circ (L_1, \ldots, L_{i-1}, L_i \cup L_{i'}, L_{i+1}, \ldots, L_n) \\ = L \circ (L_1, \ldots, L_{i-1}, L_i, L_{i+1}, \ldots, L_n) \cup L \circ (L_1, \ldots, L_{i-1}, L_{i'}, L_{i+1}, \ldots, L_n) \end{aligned}$$

□

Example 7.4. Let us consider the graded alphabet defined by $\Sigma_2 = \{f\}$, $\Sigma_1 = \{g\}$ and $\Sigma_0 = \{a, b\}$ and let E be the tree expression defined by

$$E = E_1 + E_2,$$

with $E_1 = f(a, a + b)$ and $E_2 = g(a)^{*_a} \cdot_a f(b, a)$.

Let us show how to calculate the derivative of E w.r.t. $t = g(f(b, a))$.

$$\begin{aligned} \partial_a(E) &= \partial_a(E_1) \cup \partial_a(E_2) \\ &= \{\, f(a, \varepsilon_1),\ f(\varepsilon_1, a + b) \,\} \cup \{\, g(\varepsilon_1)^{\circledast} \circ (f(b, \varepsilon_1)) \,\} \\ &= \{f(a, \varepsilon_1),\ f(\varepsilon_1, a + b),\ g(\varepsilon_1)^{\circledast} \circ (f(b, \varepsilon_1))\} \\ \partial_b(\partial_a(E)) &= \bigcup_{E' \in \partial_a(E)} \partial_b(E') \\ &= \{\, f(\varepsilon_2, \varepsilon_1),\ g(\varepsilon_1)^{\circledast} \circ (f(\varepsilon_1, \varepsilon_2)) \,\} \\ \partial_{f(b,a)}(E) &= \partial_{f(\varepsilon_1,\varepsilon_2)}(\partial_b(\partial_a(E)) \\ &= \{\, g(\varepsilon_1)^{\circledast} \,\} \\ \partial_{g(f(b,a))}(E) &= \partial_{g(\varepsilon_1)}(\partial_{f(b,a)}(E)) \\ &= \{\, g(\varepsilon_1)^{\circledast} \,\}. \end{aligned}$$

Example 7.5. Let us consider the graded alphabet defined by $\Sigma_2 = \{f\}$, $\Sigma_1 = \{g\}$ and $\Sigma_0 = \{a\}$ and let E be the tree expression defined by

$$E = (g(a)^{*_a}) \cdot_a f(f(a, a), a).$$

Let us show how to calculate the derivative of E w.r.t. $t = f(f(a, a), a)$.

$$\begin{aligned} \partial_a(E) &= \{\, g(\varepsilon_1)^{\circledast} \circ (f(f(a, \varepsilon_1), a)),\ g(\varepsilon_1)^{\circledast} \circ (f(f(a, a), \varepsilon_1), \\ &\qquad g(\varepsilon_1)^{\circledast} \circ (f(f(\varepsilon_1, a), a)) \,\} \\ \partial_a(\partial_a(E)) &= \bigcup_{E' \in \partial_a(E)} \partial_a(E') \\ &= \{\, g(\varepsilon_1)^{\circledast} \circ (f(f(\varepsilon_1, \varepsilon_2), a)),\ g(\varepsilon_1)^{\circledast} \circ (f(f(a, \varepsilon_2), \varepsilon_1)), \\ &\qquad g(\varepsilon_1)^{\circledast} \circ (f(f(\varepsilon_1, a), \varepsilon_2)),\ g(\varepsilon_1)^{\circledast} \circ (f(f(a, \varepsilon_1), \varepsilon_2)), \end{aligned}$$

$$\begin{aligned}
\partial_a(\partial_a(\partial_a(E))) &= \bigcup_{E'' \in \partial_a(\partial_a(E))} \partial_a(E'') \\
&= \{\, g(\varepsilon_1)^{\circledast} \circ (f(f(\varepsilon_2, \varepsilon_3), \varepsilon_1)),\ g(\varepsilon_1)^{\circledast} \circ (f(f(\varepsilon_1, \varepsilon_3), \varepsilon_2)), \\
&\qquad g(\varepsilon_1)^{\circledast} \circ (f(f(\varepsilon_2, \varepsilon_1), \varepsilon_3)),\ g(\varepsilon_1)^{\circledast} \circ (f(f(\varepsilon_1, \varepsilon_2), \varepsilon_3)), \\
&\qquad g(\varepsilon_1)^{\circledast} \circ (f(f(\varepsilon_3, \varepsilon_2), \varepsilon_1)),\ g(\varepsilon_1)^{\circledast} \circ (f(f(\varepsilon_3, \varepsilon_1), \varepsilon_2)) \,\} \\
\partial_{f(a,a)}(\partial_a(E)) &= \partial_{f(\varepsilon_1,\varepsilon_2)}(\partial_a(\partial_a(\partial_a(E)))) \circ (\varepsilon_1, \varepsilon_2) \\
&= \{\, g(\varepsilon_1)^{\circledast} \circ (f(\varepsilon_1, \varepsilon_4)) \circ (\varepsilon_1, \varepsilon_2) \,\} \\
\partial_{f(f(a,a),a)} &= \partial_{f(\varepsilon_1,\varepsilon_2)}(\partial_{f(a,a)}(\partial_a(E))) \\
&= \{\, g(\varepsilon_1)^{\circledast} \,\}.
\end{aligned}$$

Let us now show that the computation of an automaton from the partial derivation cannot be achieved *via* the same algorithm as the case of derivation. Let us first try to extend the previous algorithm. Given a tree expression E over an alphabet Σ, we first compute the set

$$D_0(E) = \bigcup_{a \in \Sigma_0} \partial_a(E).$$

From this set, we compute the tree automaton $A_0 = (\Sigma, D_0(E), F_0, \delta_0)$ where

$$F_0 = \{E' \in D_0(E) \mid \varepsilon_1 \in L(E')\}, \qquad \delta_0 = \{(a, E') \mid a \in \Sigma_0, E' \in \partial_a(E)\}.$$

From this step, one can choose a total function tree_0 associating any tree expression E' in $D_0(E)$ with a tree t such that

$$\mathrm{tree}_0(E') = t \Rightarrow E' \in \partial_t(E).$$

Notice that one can consider a *greedy* choice, that is a choice that minimizes the number of trees in the codomain of the function tree_0.

From this induction basis, let us consider the transition set δ_n inductively defined by

$$\begin{aligned}
\delta_n = \{((E'_1, \ldots, E'_m), f, E') \mid & E' \in \partial_{f(t_1,\ldots,t_m)}(E), t_i = \mathrm{tree}_{n-1}(E'_i), \\
& f \in \Sigma_m, E'_1, \ldots, E'_m \in D_{n-1}(E)\}.
\end{aligned} \tag{7.1}$$

Let us consider the set $D_n(E) = D_{n-1}(E) \cup \pi_3(\delta_n)$, where π_3 is the classical projection defined by $\pi_3(X) = \{z \mid (\_, \_, z) \in X\}$. Obviously, one can once again choose a total function tree_n associating any tree expression E' in $D_n(E)$ with a tree t such that

$$\mathrm{tree}_n(E') = t \Rightarrow E' \in \partial_t(E),$$

by choosing a transition $((E'_1, \ldots, E'_m), f, E')$ in δ_n for any tree expression E' in $D_n(E) \setminus (D_{n-1}(E))$ and defining t as $f(\mathrm{tree}_{n-1}(E'_1), \ldots, \mathrm{tree}_{n-1}(E'_k))$. Finally, by considering the set

$$F_n = \{E' \in D_n(E) \mid \varepsilon_1 \in L(E')\}, \tag{7.2}$$

we can define the tree automaton $A_n = (\Sigma, D_n(E), F_n, \delta_n)$.

Let us consider the expression $E = f(a,a) + f(a,b) + f(b,a)$. Then

$$\begin{aligned}\delta_a(E) &= \{f(\varepsilon_1,a), f(a,\varepsilon_1), f(\varepsilon_1,b), f(b,\varepsilon_1)\},\\ \delta_b(E) &= \{f(\varepsilon_1,a), f(a,\varepsilon_1)\},\\ D_0 &= \{f(\varepsilon_1,a), f(a,\varepsilon_1), f(\varepsilon_1,b), f(b,\varepsilon_1)\}.\end{aligned}$$

Let us choose $\mathrm{tree}_0(E') = a$ for any expression E' in D_0. By construction, the set δ_1 contains the transition $((f(\varepsilon_1,a), f(a,\varepsilon_1)), f, \varepsilon_1)$, since $\mathrm{tree}_0(f(\varepsilon_1,a)) = \mathrm{tree}_0(f(a,\varepsilon_1)) = a$ and since $\partial_{f(a,a)}(E) = \{\varepsilon_1\}$. In this case, since there also exist the transitions $(b, f(\varepsilon_1,a))$ and $(b, f(a,\varepsilon_1))$, the tree $f(b,b)$ is recognized by the automaton, that exhibits a witness of the difference with $L(E)$.

8. Web application

The computation of a derivative and partial derivative, and both the construction of a derivative tree automaton and the classical non deterministic inductive construction have been implemented in Haskell (made in Haskell, compiled in Javascript using the REFLEX PLATFORM, represented with VIZ.JS) in order to help the reader to manipulate the notions. This web application can be found here [11]. As an example, the tree expression $\neg(g(a)^{*_a}) \cdot_a f(f(a,a),a)$ of the examples can be defined from the literal input `¬(g[a]*a).af[f[a,a],a]`.

9. Conclusion and perspectives

We have shown how to compute a derivative tree automaton as a fixed point of an inductive construction when it exists. Even when it does not exist, the process can be used in order to solve the membership test (*i.e.* does a tree belong to the language denoted by a tree expression?): it is easy to see that, for a tree t of height h, the tree automaton A_h is sufficient, since $t \in L(A_h)$ if and only if $t \in L(A)$. As an example, consider the tree automaton A_1 of Example 6.6. This tree automaton is sufficient to determine that the trees in $T(\{f,g,b,c\})$ belong to $L(E)$. And even the subautomaton of A_h restricted to the transitions used in order to compute $\Delta(t)$ is enough. Moreover, due to the independence of the computations of derivatives, this process can be performed in a parallel/concurrent way.

Furthermore, we can wonder whether the choices of the tree_* functions during the computation of the derivative automaton impact the produced automaton. We conjecture that all of these choices lead to a unique automaton, and the statement of Proposition 6.1 could be replaced by $\Delta(t) = \{d_t(E)\}$.

Let us notice that this fixed point computation cannot be extended directly to deal with partial derivatives but the partial derivation can be used to solve the membership test syntactically. This is not the only loss of the extension of partial derivation from words to trees. It also seems that partial derivation tends to produce more expressions than derivation. It could be the consequence of the distributions that occur when the product and the composition are applied to sets of expressions in conjunction with the composition closure of sets of expressions, that seems to cancel the reduction power of this process.

Finally, the ACI rules of the sum used here are the same as used in the case of words, the difficulties that we have found reside on the sufficiency of these rules to prove that the set of derivatives is finite. Seen that we deal with symbols of rank ≥ 1 and with operations like the composition ($\circ$), the a-product ($\cdot_a$), the iterated composition ($\circledast$) which are somehow "complicated". However, all the examples that we have made (taking into account these rules) halt. So for now, this is only an hypothesis to justify. The study of the finiteness of the set of (similar) derivatives is the next step of our study: are the three ACI rules sufficient to obtain a finite set of derivatives? Moreover, the same question arises as far as partial derivatives are concerned: unlike the word case, does the partial derivation need reduction rules to produce a finite set of derived terms from an expression?

References

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

[2] S. Attou, L. Mignot and D. Ziadi, Extended tree expressions and their derivatives, in NCMA, Valencia, Spain, July 2–3, 2019 (2019) 47–62.
[3] J.A. Brzozowski, Derivatives of regular expressions. *J. ACM* **11** (1964) 481–494.
[4] P. Caron, J. Champarnaud and L. Mignot, Partial derivatives of an extended regular expression, in LATA, vol. 6638 of *Lecture Notes in Computer Science*. Springer (2011) 179–191.
[5] P. Caron, J. Champarnaud and L. Mignot, A general framework for the derivation of regular expressions. *RAIRO - Theor. Inf. Appl.* **48** (2014) 281–305.
[6] J.-M. Champarnaud, L. Mignot, N.O. Sebti and D. Ziadi, Bottom-up quotients for tree languages. *J. Autom. Lang. Combinat.* **22** (2017) 243–269.
[7] G.P. Huet, The zipper. *J. Funct. Program.* **7** (1997) 549–554.
[8] S. Kleene, Representation of events in nerve nets and finite automata. *Automata Studies, Ann. Math. Stud.* **34** (1956) 3–41.
[9] D. Kuske and I. Meinecke, Construction of tree automata from regular expressions. *RAIRO – Theor. Inf. Appl.* **45** (2011) 347–370.
[10] É. Laugerotte, J.-G. Luque, L. Mignot and F. Nicart, Multilinear representations of free pros. *Linear Multilinear Algeb.* (2019) 1–45.
[11] L. Mignot, Application: Bottom up derivatives. http://ludovicmignot.free.fr/programmes/BottomUpPartialDerivatives/index.html. Accessed: 2019-12-08.
[12] J.W. Thatcher and J.B. Wright, Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Syst. Theory* **2** (1968) 57–81.