N d'ordre : 142/2021-C/MT

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR
ET DE LA RECHERCHE SCIENTIFIQUE
UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE
HOUARI BOUMEDIENE
FACULTE DE MATHEMATIQUES

Thèse de Doctorat

Présentée pour l'obtention du grade de Docteur

En : Mathématiques

Spécialité : Recherche Opérationnelle et Mathématiques Discrètes (ROMaD)

par

Samira ATTOU

Titre

Expressions rationnelles d'arbres et leurs représentations par des automates compacts.

Soutenue publiquement, le **11/07/2021** à **9h:00**, devant le jury composé de :

M. Mourad BOUDHAR,	Professeur à l'USTHB,	Président
M. Hacène BELBACHIR	Professeur à l'USTHB,	Directeur de thèse
M. Ludovic MIGNOT,	Maître de Conférence (HDR) à Université de Rouen, Normandie, France	Co-Directeur de thése
Mme. Nacera BENSAOU,	Maître de Conférence à l'USTHB,	Examinatrice
Mme. Hadda CHERROUN,	Professeur à l'Université Amar Télidji Laghouat,	Examinatrice
M. Djelloul ZIADI	Professeur à Université de Rouen, Normandie, France	Invité

جامعة هواري بومدين
للعلوم والتكنولوجيا
USTHB

Regular tree expressions and their representations by compact automata*

Présenté par :

Samira ATTOU (**)

Faculté de Mathématiques

Département de Recherche Opérationnelle

Abstract: A fundamental result in the theory of formal languages asserts the equivalence between regular expressions and finite automata, each of them denote precisely the same regular languages. For practical reasons, it is important to convert a regular expression into a finite automaton. That is why a lot of algorithms have been proposed to realize this conversion.

This result is extended to tree languages. The conversion of a given regular tree expression into a tree automaton has been widely studied. However, classical interpretations are based upon a Top-Down interpretation of tree automata.

In this thesis, we propose new constructions based on Glushkov's one, on the one by Ilie and Yu and on the one by Brzozowski using a Bottom-Up interpretation. Furthermore, for the two first constructions, we exhibit a method to factorize transitions of tree automata and show that this technique is particularly interesting for these constructions. Moreover, for the third construction method, we define a new family of extended regular tree expressions 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.

One of the main goals of the Bottom-Up interpretation is to consider the links with deterministic recognizers, something which cannot be done with classical Top-Down approaches.

Finally, in order to estimate the average complexity of algorithms related to trees (determinism, minimization, etc.) it is recommended to use *random generation* of trees. For this purpose, it is useful to know the exact number of these structures. Therefore, we propose formulae to enumerate and count the number of ranked trees by size and by height.

Mots clés: trees; tree automata; tree languages; regular tree expressions; derivatives; Bottom-Up; Top-Down; enumeration; generalized Catalan numbers.

*Thèse de Doctorat LMD.

**Directeur de Thèse: Hacène BELBACHIR (Professeur, USTHB),

***Co-Directeur de Thèse: Ludovic MIGNOT (Maître de Conférences HDR, Université de Rouen Normandie).

جامعة هواري بومدين
للعلوم والتكنولوجيا
USTHB

Expressions rationnelles d'arbres et leurs représentations par des automates compacts.*

Présenté par :

Samira ATTOU (**)

Faculté de Mathématiques

Département de Recherche Opérationnelle

Résumé : Un résultat fondamental de la théorie des langages formels affirme l'équivalence entre expressions régulières et automates finis, chacun d'eux reconnaissant précisément les mêmes langages réguliers. Dans la pratique, il est important de convertir une expression régulière en un automate fini. C'est pourquoi de nombreux algorithmes ont été proposés pour réaliser cette conversion.

Ce résultat est étendu aux langages d'arbres. La conversion d'une expression d'arbre régulière donnée en un automate d'arbre a été largement étudiée. Cependant, les interprétations classiques sont basées sur une interprétation Top-Down des automates d'arbres.

Dans cette thèse, nous proposons de nouvelles constructions d'automates d'arbres basées sur celle de Glushkov, sur celle d'Ilie et Yu et sur celle de Brzozowski en utilisant une interprétation Bottom-Up. De plus, pour les deux premières constructions, nous présentons une méthode de factorisation des transitions d'automates d'arbres et montrons que cette technique est particulièrement intéressante pour ces constructions. De plus, pour la troisième méthode, nous définissons une nouvelle famille d'expressions d'arbres régulières étendues et nous montrons comment calculer un automate d'arbre inductif de type Brzozowski ; le point fixe de cette construction, lorsqu'elle existe, est l'automate d'arbre des dérivées. Un tel automate d'arbre déterministe peut être utilisé pour résoudre efficacement le test d'appartenance. L'un des principaux objectifs de l'interprétation ascendante est de considérer les liens avec les reconnaisseurs déterministes, chose qui ne peut pas être réaliser avec les approches descendantes classiques. Enfin, afin d'estimer la complexité moyenne des algorithmes liés aux arbres (déterminisme, minimisation, etc.), il est recommandé d'utiliser *la génération aléatoire* d'arbres. Pour cela, il est utile de connaître le nombre exact de ces structures. Par conséquent, nous proposons des formules pour énumérer et compter le nombre d'arbres rangés par taille et par hauteur.

Mots clés : arbres ; automates d'arbres ; langages d'arbres ; expression rationnelle d'arbre ; derivation ; Bottom-Up ; Top-Down ; enumeration ; nombres de Catalan généralisés.

*Thèse de Doctorat LMD.

**Directeur de Thèse : Hacène BELBACHIR (Professeur, USTHB),

***Co-Directeur de Thèse : Ludovic MIGNOT (Maître de Conférences HDR, Université de Rouen Normandie).

Contents

List of Figures

List of Tables

I dedicate this thesis to my parents.
To my dad who gave me the passion
and the support to do research.
To my mother for her endless love
and tenderness.

Acknowledgements

This thesis is the result of the support of several people to whom I am extremely thankful.

First, I am extremely and deeply grateful to my supervisors, Prof Hacène BELBACHIR and Mr Ludovic MIGNOT for their patience, their treasured support, their valuable advice and their encouragement during my PhD study and research. Thank you for giving me the opportunity to discover such a magnificent research field.

Besides my supervisors, I would like to acknowledge the rest of my thesis jury members: Prof Mourad BOUDHAR, Madame Nacera BENSAOU and Prof Hadda CHERROUN for accepting to evaluate my thesis.

I would like to offer my special thanks to Prof Djelloul ZIADI for his collaboration and his support. Thank you for giving me the motivation to do research.

My thanks to all my professors of the department of Operation Research of the USTHB. I had such an honor to be your student. Thank you for all what you have taught me.

I would like to express my sincere gratitude to all the members of RECITS laboratory (especially CATI team) of the department of Operation Research of the USTHB and to all the members of GR²IF laboratory of the department of Computer Science of the University of Rouen Normandy who welcomed me, who helped me and advanced my research work and to express my satisfaction and my gratitude for having been able to work in good conditions in a very pleasant environment.

I would also like to say a heartfelt thank you to Mummy and Daddy for always believing in me and encouraging me. Thank you for making me that ambitious girl.

A very special thank to my best friend Asma ABIDI for always being there for me no matter the distance separating us. I am grateful too to my friends Azhar HAMANA and Nadia BOUSSAHA, i would never forget your help.

Finally I express my gratitude to the people whose names that would not appear on this page that have helped me in one way or another. Thank you to everyone for everything.

1 Introduction

It was in 1936 that Alan Turing invented the first abstract machine (an automaton) called *Turing Machine* [63]. He claimed that this machine is universal *i.e.* any algorithm can be computed by a Turing Machine. Due to Turing's ideas, John von Neumann could construct in 1945 the architecture of a real machine: the computer [65].

Since 1950, researchers and computer scientists started realizing programming languages and the computer softwares that implement them. To do so, they needed tools to read, analyze, execute instructions and also to verify whether the input program is compatible with the syntax of a given programming language. Such tools are called *interpreters* and *compilers*. For this reason, an abstract machine less complicated and less universal than Turing machine called *a finite automaton* (or finite state machine) has been invented.

Finite automata are recognizers used in various domains of applications: in electronics [8, 34], in linguistics [21], in algebra [54] and especially in computer science, *e.g.* to represent (finite or infinite) languages, or to solve the membership test, *i.e.* to verify whether a given element belongs to a language or not.

In 1956, Chomsky introduced the model of *Chomsky hierarchy* [21]. He proposed a classification of four types of formal languages, he associated to each type of languages an automaton that recognizes it. As an example, Turing machines are the most complicated machines. They are the ones defining *recursively enumerable languages*. Finite automata are the simplest abstract machines that recognize regular languages.

Five years ago before, another mathematical object called *regular expression* has been created by Kleene [38]. This object represents a string that denotes a set of strings. Kleene showed how the human nervous system works using regular expression contrary to McCulloch and Pitts when they presented the same model in 1943 using finite automata [44].

Regular expressions are used to describe formal languages, to manipulate programming languages especially in the compilation phase. They also can be found in text editors like *Ed* created by Thompson in 1968 [62].

A fundamental theorem in automata theory has been proposed by Kleene [37] in 1956. He asserts the existence of an equivalence between the language accepted by an automaton and the language denoted by regular expressions and that these latter are a compact representation for finite automata.

The powerful of this theorem resides on the possibility of presenting an infinite object (infinite language) by two equivalent finite objects.

The proof of this theorem is given by proposing conversions algorithms either from automata to regular expressions [3, 14, 46] or from regular expression to finite automata [62, 28, 2, 35].

In our study, we will focus on the conversion of a regular expression to a finite state machine in order to recognize languages. For example:

Glushkov [28] showed how to construct a nondeterministic finite automaton with $n+1$ states where n represents the number of occurrences of symbols of a given regular expression. The main idea of the construction is to define some particular sets named First (containing the symbols starting a word in the denoted languages), Follow (containing the symbols following a given symbol in the words in the denoted languages) and Last (containing the symbols ending a word in the denoted languages) that are computed with respect to

the occurrences of the symbols that appear in the expression. These so-called Glushkov automata (or Position automata) are finite state machines that have been deeply studied. They have been structurally characterized by Caron and Ziadi [15], who could invert the Glushkov computation by constructing an expression with n symbols from a Glushkov automaton with $n+1$ states. They have been considered too in the theoretical notion of one-unambiguity by Bruggemann-Klein and Wood [12], characterizing regular languages recognized by a deterministic Glushkov automaton, or for practical applications, like expression updating [10]. Finally, it is also related to combinatorial research topics. As an example, Nicaud [51] proved that the average number of transitions of Glushkov automata is linear. Moreover, the Glushkov automata can be easily reduced into the Follow automata [35] in the case of word by applying an easy-to-compute congruence from the position functions.

Another method has been proposed to provide the conversion of a given regular expression to a finite word automaton (appeared in 1964) was Brzozowski's construction [13]; its basic 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.

And last, Antimirov [2], 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).

Notice that these constructions are used to define words. However, there exists another class more generalized than words called *trees* which have two ways to read them: from root to leaves (Top-Down) or from leaves to the root (Bottom-Up).

Donald Knuth described them as the most important nonlinear structures that appear in computer science [40]. This structure attracted a lot of researchers, mathematicians and computer scientists considering the fact that it allows presenting, storing and organizing information (data) [60, 39] in a hierarchical way as used in XML documents. In mathematics, especially in graph theory, counting trees was one of the most frequently treated problem. Cayley [16] was the first who gave the enumeration of labeled trees.

Thatcher was one of the researchers interesting on the notion of trees. He extend the theory of word automata to tree automata theory [61].

So, one can wonder: can we extend all the studies related to word automata to the case of tree automata? More precisely, is there an equivalence between tree automata and regular tree expressions recognizing the same language? Can we transform a regular tree expression to a tree automaton recognizing the same language? If yes, how can we prove it such that we have two ways to interpret trees?

The answer of these questions is YES. Kleene theorem still hold in the case of trees. Many algorithms have been proposed:

- The Glushkov construction was extended to tree automata [43, 49] by using a Top-Down interpretation of regular tree expressions.
- Antimirov construction has been extended to tree automata. Kuske and Meinecke [42], in 2011, introduced an algorithm to convert a regular tree expression into a non-deterministic tree automaton in a Top-Down interpretation.

The Top-Down interpretation can be problematic while considering determinism. Indeed, it is well-known that there exist regular tree languages that cannot be recognized by

Top-Down deterministic tree automata [23]. For instance, a deterministic tree automata does not recognize the language $\{f(a,b), f(b,a)\}$.

To solve this limit, we propose throughout this thesis algorithms based on the Bottom-Up interpretation.

Moreover, in order to estimate and analyze the average complexity of tree algorithms (*e.g.* conversion methods [30], minimizing [29], determinism, etc.), it is recommended to use *random generation* of ranked trees as used in [20, 50] in the study of deterministic and non deterministic finite automata (case of unary trees *i.e.* words), and to do so, it is useful and important to know the exact number of these structures.

This thesis contains five chapters in total and they are divided into two parts: the first part containing four chapters, we propose three new constructions of tree automata: the first based on the construction of Glushkov, the second is based on the one of Ilie and Yu and the third is based on the notion of derivation of Brzozowski of an extended tree expression. Our constructions are given in a Bottom-Up way to solve “somehow" the limit of the Top-Down interpretation.

The second part of this thesis contains one chapter, we propose on it an enumeration of some ranked trees over a finite set of symbols w.r.t. some conditions.

Part I

Constructions of Tree Automata

2 Preliminaries

In this chapter, we will introduce the tools that we will deal with in this thesis. We will give notations, definitions and notions related to tree automata.

In Section 2.1, we introduce the structure of *trees* and the associated operations. The set of *tree languages* and its associated operations will be shown in Section 2.2. Section 2.3 is reserved to introduce the notion of *tree automata*. Last, we present in Section 2.4 the *regular tree expression*.

The content of this chapter is extracted from [19, 22, 23].

2.1 Trees

The structure of trees is the structure the most used and the most important in computer science. Donald Knuth described *trees* as the most important nonlinear structures that appear in computer science contrary to arrays and lists [40]. This structure attracted a lot of researchers, mathematicians and computer scientists considering the fact that it allows presenting and storing information (data) in a hierarchical way. In *tree automata theory*, we distinguish two kinds of trees, ranked and unranked trees [23]. Throughout this thesis, we only deal with ranked trees.

2.1.1 Ranked Trees

Definition 2.1.1. A ranked alphabet $(\Sigma, arity)$ *is a couple constituted of:*

- An alphabet Σ *which contains a finite set of distinct symbols,*
- *An* arity *(or rank) is a mapping defined by* $\text{arity} : \Sigma \to \mathbb{N}$ *allowing to associate to each symbol of* Σ *a rank.*

We denote by $(\Sigma_n)_{n\geq 0}$ the set of symbols of rank n. A symbol $f \in \Sigma_k$ (or $f^k \in \Sigma$) is said to be of arity k. The set Σ_0 contains symbols of arity zero named *constants*. We denote by $\Sigma_{\geq 1}$ the set of symbols of rank greater or equal to one.

Definition 2.1.2. A ranked tree t *over* Σ *is constituted of root symbol, internal nodes and leaves. The rank of the symbols labelling a node of a tree* t *represents the number of children (sons) of these symbols and the symbols of rank zero are labelled in the leaves. The empty tree* ε *is of rank one. Formally, a ranked tree* t *is inductively defined by*

$$t = \begin{cases} a & \text{if } a \in \Sigma_0, \\ f(t_1, \ldots, t_k) & \text{if } f \in \Sigma_k, \end{cases}$$

where t_i are *subtrees* of t for $1 \leq i \leq k$. We denote this relation by $t_i \preceq t$. The set T_Σ is the set containing all the ranked trees over Σ.

Let us see some properties related to ranked trees:

- $\text{root}(t)$ is the root symbol of the tree t, *i.e.*

$$\text{root}(f(t_1, \ldots, t_k)) = f. \tag{2.1}$$

- $k - child(t)$ is the root of the k^{th} subtree if it exists, *i.e.*

$$k - child(f(t_1, \ldots, t_k)) = root(t_k). \tag{2.2}$$

- $Leaves(t)$ is the set containing the leaves of a tree t, *i.e.*

$$Leaves(t) = \{a \in \Sigma_0 \mid a \preceq t\}. \tag{2.3}$$

- $k-$ary trees are the trees containing k missing leaves or the trees having k apparitions of ε's.

We can present the tree t graphically as follows

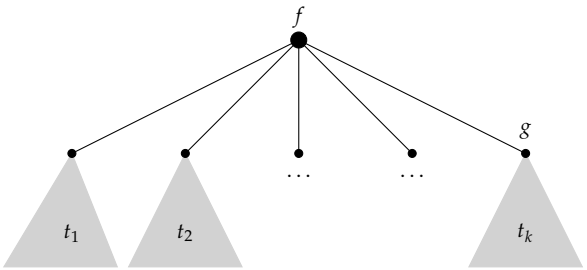


FIGURE 2.1: Illustration of the tree $t = f(t_1, \ldots, t_k)$.

Example 2.1.1. *Let us consider a ranked alphabet defined by* $\Sigma = \{f^3, g^2, a, b, c\}$. *The set of trees over* Σ *is given by*

$$T_\Sigma = \{f(g(a,b),c,a),\ g(a,b),\ f(a,b,g(c,c)),\ g(f(a,a,b),c),\ f(g(a,b),c,g(b,a)),\ldots\}.$$

Let us consider a tree $t = f(g(a,b),c,g(b,a)) \in T_\Sigma$.

- *The root and the leaves of t are*

$$root(t) = f, \qquad Leaves(t) = \{a, b, c\}.$$

- *The subtrees of t are*

$$\begin{aligned} f(g(a,b),c,g(b,a)) &\preceq t, & g(a,b) &\prec t,\\ c &\prec t, & g(b,a) &\prec t. \end{aligned}$$

- *The* $k - child$ *of t are*

$$\begin{aligned} 1 - child(t) &= root(g(a,b)), & 2 - child(t) &= root(c),\\ &= g, & &= c,\\ 3 - child(t) &= root(g(b,a)),\\ &= g. \end{aligned}$$

The tree t is presented in Figure 2.2

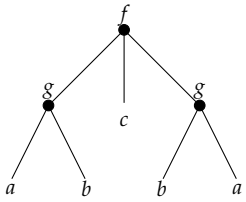


FIGURE 2.2: A tree $t = f(g(a,b),c,g(b,a))$.

2.1.2 Operations over Trees

Due to the recursive structure of trees, we can apply and associate operations to trees in order to produce another trees. For instance, we may concatenate a tree in another tree or eliminate a subtree from a tree and so on. Let us see some of these actions:

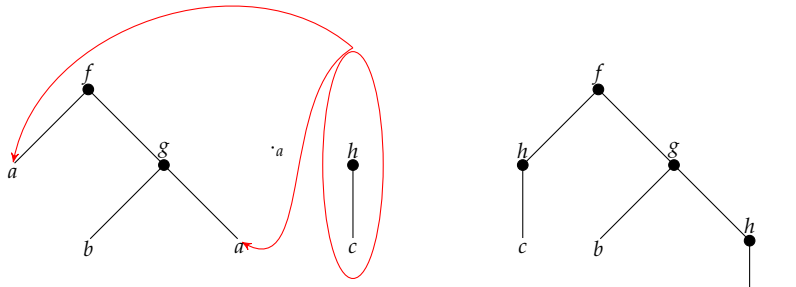
The Concatenation

Definition 2.1.3. *Let $t, s \in T_\Sigma$ and $a \in \Sigma_0$. We denote by $t \cdot_a s$ the tree obtained by substituting the tree s in the occurrences of the symbol a in the tree t and it is defined by*

$$t \cdot_a s = \begin{cases} s & \text{if } t = a, \\ f(t_1 \cdot_a s, \ldots, t_k \cdot_a s) & \text{if } t = f(t_1, \ldots, t_k), \end{cases}$$

As an example, let us consider the trees $t = f(a, g(b,a))$ and $s = h(b)$ in T_Σ, where $f, g \in \Sigma_2$ and $a, b, c \in \Sigma_0$. Then

$$f(a, g(b,a)) \cdot_a h(c) = f(h(c), g(b, h(c)))$$



$f(a, g(b,a)) \cdot_a h(c)$ $\qquad$ $f(h(c), g(b, h(c)))$

The Composition

Definition 2.1.4. *Let $t = f(t_1, \ldots, t_k)$ and let m trees $t'_1, \ldots, t'_m$ in T_Σ. The composition of trees is the operation of placing (substituting) the trees t'_i (for $1 \leq i \leq m$) in the $i-$th missing leaf of t* (i.e. *in each ε_{x_i} that appears in t). We denote this operation by $t \circ (t'_1, \ldots, t'_m)$ and it is inductively defined by*

$$t \circ (t'_1, \ldots, t'_m) = \begin{cases} t'_1 & \text{if } t = \varepsilon_1, \\ f((t_j \circ (t'_i)_{x_i \in Ind_\varepsilon(t_j)})_{1 \leq j \leq k}) & \text{if } t = f(t_1, \ldots, t_k), \end{cases}$$

where $Ind_\varepsilon(t) = \{x_1, \ldots, x_m\}$ ($x_j \in \mathbb{N} \setminus \{0\}$). This set is an ordered set containing the ε-indices that appear in the k-ary tree t. The set of ε-indices is used to guarantee grafting

each tree to the right place in the tree t, in other words, the indices define the order of composition.

Formally a k-ary tree t with $R = Ind_\varepsilon(t)$ is defined inductively by

- $t = \varepsilon_j$ $(j \in \mathbb{N} \setminus \{0\})$ in this case $k = 1 \wedge R = \{j\}$,
- $t = f(t_1, \ldots, t_k)$ ($f \in \Sigma_k$ and for $1 \leq j \leq k$ the t_j is a n_j-ary tree with R_j ε-indices, such that $1 \leq j \leq j' \leq k$, $R_j \cap R'_j = \emptyset$, in this case $k = \sum_{1 \leq j \leq k} n_j$ and $R = \bigcup_{1 \leq j \leq k} R_j$.

We denote by $T(\Sigma)_k$ the set of k-ary trees over Σ.

As an example, imagine that the ε of a given k-ary tree are not indexed, let us consider the composition of a given ternary tree by (a, b, c) then,

$$f(\varepsilon, g(\varepsilon, \varepsilon)) \circ (a, b, c) = f(a, g(b, c))$$

or

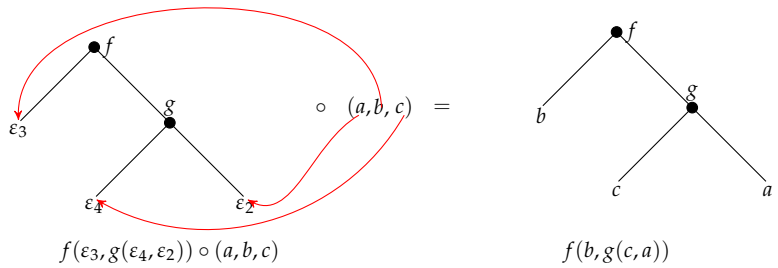
$$g(\varepsilon, f(\varepsilon, \varepsilon)) \circ (b, c, a) = g(b, f(c, a)).$$

Notice that there is no order to respect. Furthermore, what if we make a composition of a set of k-ary trees by a list of trees. For instance, $\{f(\varepsilon, g(\varepsilon, \varepsilon)), g(\varepsilon, f(\varepsilon, \varepsilon))\} \circ (a, b, c)$ in this case, we will not obtain a list containing $\{f(a, g(b, c)),\ g(b, f(c, a))\}$. For this reason, we introduced the ε-indices set.

Let us consider $f(\varepsilon_3, g(\varepsilon_4, \varepsilon_2)) \circ (a, b, c)$, where $f \in \Sigma_2$, and $a, b, c \in \Sigma_0$.

We have

$$Ind_\varepsilon(f(\varepsilon_3, g(\varepsilon_4, \varepsilon_2))) = \{x_1 = 2, x_2 = 3, x_3 = 4\}$$



The Partial Composition

Definition 2.1.5. *The partial composition $\circ_1$ is defined for any k-ary tree t of ε-indices $\{j_1, \ldots, j_k\}$ with $k \geq 1$ and for any tree t' by*

$$t \circ_1 t' = t \circ (t', (\varepsilon_l)_{j_2 \leq l \leq k_k})$$

For example, let us consider $f(\varepsilon_1, g(\varepsilon_2, a)) \circ_1 h(a)$, where $f, g \in \Sigma_2$, $h \in \Sigma_1$ and $a \in \Sigma_0$. Then

$$f(\varepsilon_1, g(\varepsilon_2, a)) \circ_1 h(a) = f(\varepsilon_1, g(\varepsilon_2, a)) \circ (h(a), \varepsilon_2) = f(\varepsilon_1, g(h(a), a)).$$

The Quotient

Definition 2.1.6. *Let t be a k-ary tree and t' be a k'-tree. The quotient of a tree t w.r.t. t' is the operation of deleting nodes in trees, denoted by $t'^{-1}(t)$. This operation allows obtaining a set of trees t'' where each tree contains an occurrence of ε.*

In the case where an ε_x appears in t, we should make a reindexing of $\varepsilon-$indices and then increment them. Given an integer z, we denote by $Inc_\varepsilon(z, t)$ the substitution of ε_x by ε_{x+z}.

Formally, the trees t'' should satisfy these conditions:

$$t'' \circ (t', (\varepsilon_{x_z})_{1\leq z\leq k-k'}), \qquad Ind_\varepsilon(t'') = \{1, (x_z+1)_{1\leq z\leq k-k'}\} \tag{2.4}$$

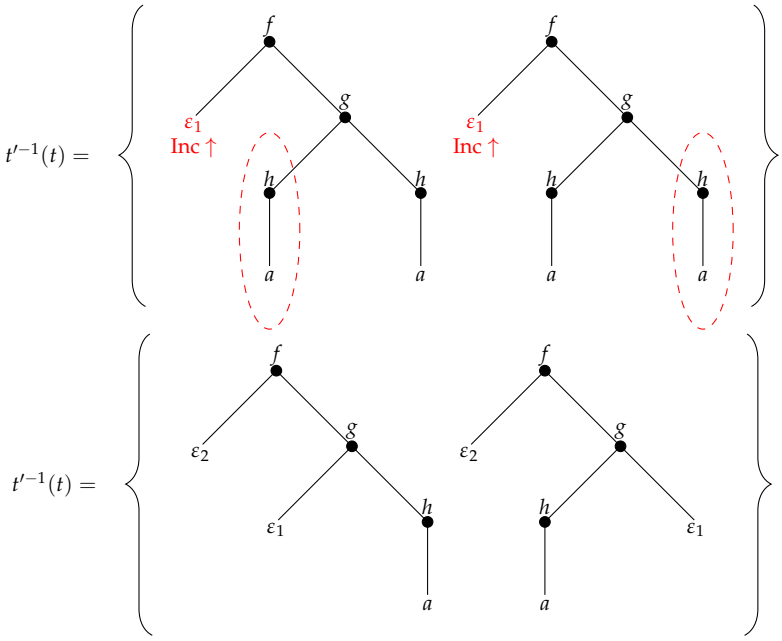
Consequently,

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

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

According to [23], the obtained tree is called *context*. As an example, let us consider the unary tree $t = f(\varepsilon_1, g(h(a), h(a)))$ with $Ind_\varepsilon(t) = \{x_1 = 1\}$ and the nullary tree $t' = h(a)$ in T_Σ. Then,

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



Notice that

$$t'' \circ (h(a), \varepsilon_1) = f(\varepsilon_2, g(h(a), h(a))), \quad Ind_\varepsilon(t'') = \{1, x_1 + 1\} = \{1, 2\}.$$

Consequently, $t'' \circ (t', \varepsilon_1) = t$. Thus, the operation of composition is the dual operation of the quotient.

2.2 Tree Languages

In this section, we define the tree languages and the operations that we deal with.

Definition 2.2.1. *A tree language L is a subset of T_Σ, contains a finite or an infinite set of trees. The set of tree languages is denoted by $L(\Sigma)$.*

Example 2.2.1. *Let us consider the ranked alphabet and the set of ranked trees defined in Example 2.1.1. Let us consider only the trees (the language $L_1 \subseteq T_\Sigma$) in which f is the root of.*

$$L_1 = \{f(a,b,c),\ f(c,b,a),\ f(g(a,b),c,a),\ f(a,b,g(c,c),\ f(g(a,b),c,g(b,a)),\ f(a,f(b,b,b),c)\ldots\}.$$

Let us consider the set of binary trees $L_2 \subseteq T_\Sigma$.

$$L_2 = \{g(a,b),\ g(a,a),\ g(b,b)\ g(b,a),\ g(g(a,a),c),\ g(g(c,b),g(a,c)),\ g(g(g(a,g(a,c)),b),a),\ldots\}.$$

As in the case of words, this set of tree language deals with operations. Let us define each operation apart. For any integer $k \geq 0$, for any k languages $L_1, \ldots, L_k \subseteq T_\Sigma$, for any $f \in \Sigma_k$

$$f(L_1, \ldots, L_k) = \{f(t_1, \ldots, t_k) \mid t_i \in L_i, 1 \leq i \leq k\}.$$

- *The tree substitution $t \cdot_a L$* is the language obtained by substituting the occurrences of a by the language L in a tree t and it is inductively defined by:

$$t \cdot_a L = \begin{cases} L & \text{if } t = a, \\ c & \text{if } c \in \Sigma_0 \setminus \{a\}, \\ f(t_1 \cdot_a L, \ldots, t_k \cdot_a L) & \text{otherwise.} \end{cases}$$

- *The $a-$product $\cdot_a$* (or $a \leftarrow$) is the language obtained by substituting the symbol a of the trees t_1 by a set of trees t_2 *i.e.*

$$L_1 \cdot_a L_2 = \{t_1 \cdot_a L_2 \mid t_1 \in L_1\}.$$

- *The iterated $a-$product ^{a,n}* is a sequence of successive iterations of the $a-$product and it is inductively defined by

$$L^{a,n} = \begin{cases} \{a\}, & \text{if } n = 0, \\ L \cdot_a L^{a,n-1} \cup L^{a,n-1} & \text{otherwise.} \end{cases}$$

  As an example, let us consider the language defined by $L = \{g(a,b)\}$.

$$L^{a,0} = \{a\} \qquad L^{a,1} = \{a\} \cup \{g(a,b)\} \cdot_a \{a\} = \{a,\ g(a,b)\}$$

$$\begin{aligned} L^{a,2} &= L^{a,1} \cup \{g(a,b)\} \cdot_a L^{a,1} \\ &= L^{a,1} \cup \{g(a,b),\ g(g(a,b),b)\} \\ &= \{a,\ g(a,b),\ g(g(a,b),b)\}. \end{aligned}$$

- *The a-closure ^{*_a}* is the generalization of Kleene star and it is defined by

$$L^{*_a} = \bigcup_{k \geq 0} L^{a,k}.$$

  As an example, let us consider the language $L = \{g(a,b)\}$.

$$\begin{aligned} L^{*_a} &= L^{a,0} \cup L^{a,1} \cup L^{a,2} \cup \ldots \cup L^{a,k} \\ &= \{a,\ g(a,b),\ g(g(a,b),b),\ g(g(g(a,b),b),b),\ldots\}. \end{aligned}$$

Definition 2.2.2. *A tree language L is* homogeneous *if all the trees it contains has the same arity with same ε-indices. It is k-*homogeneous *if it is homogeneous and it contains only the $k-$ary trees. The set of $\varepsilon-$indices is denoted by $Ind_\varepsilon(L)$. We denote by $L(\Sigma)_k$ the set of k-homogeneous tree languages.*

Example 2.2.2. *Let us consider the ranked alphabets* $\Sigma = \{f^3, g^2, a, b, c\}$. *Let us see some examples of k-homogeneous languages.*

$$L_1 = \{a,\ b,\ c,\ g(a,b),\ f(c,a,b),\ f(g(a,b),a,c),\ g(f(a,a,a),c),\ f(b,a,g(c,c)),\ldots\}.$$
$$L_2 = \{\varepsilon_1,\ ,f(\varepsilon_1,a,b),\ g(a,\varepsilon_1),\ g(g(b,\varepsilon_1),c),\ f(b,c,\varepsilon_1),\ f(g(\varepsilon_1,c),a,b),\ldots\}$$

L_1 *is 0-homogeneous language and* L_2 *is* 1*-homogeneous language.*

As in the case of words and the case of tree languages, this kind of sets deal with operations such as:

- *The union* of two k-homogeneous languages of ε-indices R produces a k-homogeneous tree language with ε-indices R.
- *The composition* $\circ$ is extended from trees to set of trees, defined by

  $$L \circ (L_1, \ldots, L_k) = \{t \circ (t_1, \ldots, t_k) \mid t \in L, t_i \in L_i, i \leq k\},$$

  where $L \in L(\Sigma)_k$ and $L_1, \ldots, L_k \in L(\Sigma)$ such that $Ind_\varepsilon(L_i) \cap Ind_\varepsilon(L_j) = \emptyset$ (for $1 \leq i < j \leq k$). If the tree language L_i is n_i-homogeneous then the obtained tree language by the composition $L \circ (L_1, \ldots, L_k)$ is $\sum_{1 \leq i \leq k} n_i$-homogeneous.
- *The partial composition* $\circ_1$ is extended too from trees to tree languages, defined for any k-homogeneous language L of ε-indices $\{j_1, \ldots, j_k\}$ with $k \geq 1$ and for any tree language $L' \in L(\Sigma)$ by

  $$L \circ_1 L' = L \circ (L', (\varepsilon_l)_{j_2 \leq l \leq k_k}).$$
- *The iterated composition* $n_\circ$ (for $n \in \mathbb{N}$) is defined recursively by

  $$L^{n_\circ} = \begin{cases} \{\varepsilon_j\}, & \text{if } n = 0, \\ L^{n-1_\circ} \cup L^{n-1_\circ} \circ L & \text{otherwise.} \end{cases}$$

  where L is $1-$homogeneous tree language of $\varepsilon-$index$= \{j\}$.
- *The closure composition* $^\circledast$ is the extended operation of Kleene star, applied to $k-$homogeneous languages, it can be seen as the action of substituting, in each apparition of ε_1, the $1-$homogeneous language L. It is defined by

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

Example 2.2.3. *Let us consider the three* $1-$*homogeneous tree languages* $L = \{g(a, \varepsilon_1)\}$, $L_1 = \{f(a, \varepsilon_1, b)\}$ *and* $L_2 = \{\varepsilon_1\}$.

$$L \circ (L_1, L_2) = \{g(a, \varepsilon_1)\} \circ (\{f(a, \varepsilon_1, b),\ \varepsilon_1\}) = \{g(a, f(a, \varepsilon_1, b)),\ g(a, \varepsilon_1)\}.$$

$$L^{0_\circ} = \{\varepsilon_1\}, \qquad L^{1_\circ} = \{\varepsilon_1\} \cup \{\varepsilon_1\} \circ g(a, \varepsilon_1) = \{\varepsilon_1,\ g(a, \varepsilon_1)\}.$$

$$L^{2_\circ} = \{\varepsilon_1,\ g(a, \varepsilon_1)\} \cup \{\varepsilon_1,\ g(a, \varepsilon_1)\} \circ \{g(a, \varepsilon_1)\} = \{\varepsilon_1,\ g(a, \varepsilon_1),\ g(a, g(a, \varepsilon_1))\}$$

$$\begin{aligned} L^{\circledast} &= L^{0_\circ} \cup L^{1_\circ} \cup L^{2_\circ} \cup L^{3_\circ} \dots \\ &= \{\varepsilon_1,\ g(a,\varepsilon_1),\ g(a,g(a,\varepsilon_1)),\ g(a,g(a,g(a,\varepsilon_1)))\dots\}. \end{aligned}$$

2.2.1 Bottom-Up Quotient of Tree Languages

In this part, we recall the Bottom-Up quotient formulas for languages shown in [19]. The contents of this part will help to understand the results of Chapter 5.

As we have seen before, the quotient is the operation that allows deleting nodes. Quotienting in a Bottom-Up way is deleting nodes from leaves to root. An internal node (including a root) is deleted if and only if its sons are deleted.

Definition 2.2.3. *([19]) The* Bottom-Up quotient $t^{-1}(L)$ *of a tree language L w.r.t. a tree t is the tree language* $\bigcup_{t' \in L} t^{-1}(t')$.

The computation of the Bottom-Up quotient is given inductively as follows: let k be an integer, let $\alpha \in \Sigma_k$:

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

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 $1 \le z \le n$, t'_z is the tree $\mathrm{Inc}_\varepsilon(1,t_z)$.

Let L be a k-homogeneous language with $\mathrm{Ind}_\varepsilon(L) = \{j_1,\dots,j_k\}$ and j be an integer:

$$\varepsilon_j^{-1}(L) = \begin{cases} L \circ (\varepsilon_{j_1+1},\dots,\varepsilon_{j_{z-1}+1},\varepsilon_1,\varepsilon_{j_{z+1}+1},\dots,\varepsilon_{j_k+1}) & \text{if } j = j_z \in \mathrm{Ind}_\varepsilon(L), \\ \emptyset & \text{otherwise.} \end{cases} \tag{2.5}$$

As an example: let us consider a 2-ary tree $t = f(\varepsilon_2, f(\varepsilon_1, a)) \in L$ with $Ind_\varepsilon = \{j_1 = 2, j_2 = 1\}$, $f \in \Sigma_2$ and $a \in \Sigma_0$. Let us calculate $\varepsilon_2^{-1}(t)$. Then

$$\begin{aligned} \varepsilon_2^{-1}(f(\varepsilon_2,f(\varepsilon_1,a))) &= \{f(\varepsilon_2,f(\varepsilon_1,a)) \circ (\varepsilon_1,\varepsilon_{j_2+1})\} \\ &= \{f(\varepsilon_1,f(\varepsilon_2,a))\}. \end{aligned}$$

Let u be a n-ary tree with $Ind_\varepsilon(u) = \{x_1,\dots,x_n\}$. Let $t = f(t_1,\dots,t_k)$ be a l-ary tree. Let $\{y_1,\dots,y_{n-l}\} = Ind_\varepsilon(u) \setminus Ind_\varepsilon(t)$ and for all $1 \le j \le k$, $t'_j = Inc_\varepsilon(k-j,t_j)$. The Bottom-Up quotient of the tree u w.r.t. t is given by

$$t^{-1}(u) = (f^{-1}({t'_1}^{-1}(\cdots({t'_k}^{-1}(u))\cdots)) \circ (\varepsilon_1, (\varepsilon_{y_z+1})_{1\le z\le n-l}), \tag{2.6}$$

Example 2.2.4. *Let us consider a tree $u = f(\varepsilon_2, f(a,a))$ with $Ind_\varepsilon(u) = \{x_1 = 2\}$ and $t = f(a,a)$ with $f \in \Sigma_2$ and $a \in \Sigma_0$. Let us calculate $t^{-1}(u)$. Then*

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

Let us see how to compute the Bottom-Up quotient for the operations over languages:

- *The Bottom-Up quotient for the union of languages* can be computed as follows: Let t be a tree in $T(\Sigma)$, L_1 and L_2 be two languages over Σ. Then:

$$t^{-1}(L_1 \cup L_2) = t^{-1}(L_1) \cup t^{-1}(L_2). \tag{2.7}$$

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

$$t^{-1}(L) = (f^{-1}({t'_1}^{-1}(\cdots({t'_k}^{-1}(L))\cdots)) \circ (\varepsilon_1, (\varepsilon_{y_z+1})_{1\leq z\leq n-l}). \tag{2.8}$$

- *The Bottom-Up quotient for the b-product of languages* can be computed as follows: Let Σ be an alphabet. Let t be k-ary tree, L be a 0-homogeneous language, $a \in \Sigma$ and $b \in \Sigma_0$. Then:

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

  Since $L_1 \cdot_b L_2 = \bigcup_{t_1 \in L_1} t_1 \cdot_b L_2$, then:

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

  where L_1 is a k-homogeneous language, L_2 is a 0-homogeneous language, and $\circ_1$ is the partial composition defined by $L \circ_1 L' = L \circ (L', (\varepsilon_l)_{j_2 \leq l \leq j_k})$ with $\mathrm{Ind}_\varepsilon(L) = \{j_1, \ldots, j_k\}$. For example, let $L = \{b\}$ be a 0-homogeneous language, let t be a unary tree $t = f(a, f(\varepsilon_1, a), b)$, with $f \in \Sigma_3$, $\{a, b\} \in \Sigma_0$. Let us calculate $b^{-1}(t \cdot_a L)$:

$$\begin{aligned} b^{-1}(t \cdot_a L) &= b^{-1}(t) \cdot_a L \cup (a^{-1}(t) \cdot_a L) \circ_1 b^{-1}(L) \\ &= b^{-1}(f(a, f(\varepsilon_1, a), b)) \cdot_a \{b\} \cup (a^{-1}(f(a, f(\varepsilon_1, a), b)) \cdot_a L) \circ_1 b^{-1}\{b\} \\ &= \{f(a, f(\varepsilon_2, a), \varepsilon_1)\} \cdot_a \{b\} \cup (\{f(\varepsilon_1, f(\varepsilon_2, a), b),\ f(a, f(\varepsilon_2, \varepsilon_1), b)\}) \circ_1 \varepsilon_1 \\ &= \{f(b, f(\varepsilon_2, b), \varepsilon_1)\} \cup \{f(\varepsilon_1, f(\varepsilon_2, a), b),\ f(a, f(\varepsilon_2, \varepsilon_1), b)\} \\ &= \{f(b, f(\varepsilon_2, b), \varepsilon_1),\ f(\varepsilon_1, f(\varepsilon_2, a), b),\ f(a, f(\varepsilon_2, \varepsilon_1), b)\} \end{aligned}$$

- *The Bottom-Up quotient for the composition of trees* can be computed as follows: Let Σ be an alphabet. Let t be a k-homogeneous language with $\mathrm{Ind}_\varepsilon(t) = \{j_1, \ldots, j_k\}$, let $t_1, \ldots, t_k$ be k trees and $a \in \Sigma_n$. Then:

$$\begin{aligned} a^{-1}(t \circ (t_1, \ldots, t_k)) = &\bigcup_{1\leq j\leq k} t \circ ((\mathrm{Inc}_\varepsilon(1, t_l))_{1\leq l\leq j}, a^{-1}(t_j), (\mathrm{Inc}_\varepsilon(1, t_l))_{j+1\leq l\leq k}) \\ &\cup \begin{cases} a((\varepsilon_{j_{p_l}})_{1\leq l\leq n})^{-1}(t) \circ (\varepsilon_1, (\mathrm{Inc}_\varepsilon(1, t_l))_{1\leq l\leq k | \forall z, l\neq p_z})) \\ \quad \text{if } \forall 1 \leq l \leq n, \exists 1 \leq p_l \leq k, \varepsilon_l \in L_{p_l} \\ \emptyset \quad \text{otherwise.} \end{cases} \end{aligned} \tag{2.9}$$

Since $L \circ (L_1, \ldots, L_k) = \bigcup_{t \in L, (t_1, \ldots, t_k) \in L_1, \ldots, L_k} t \circ (t_1, \ldots, t_k)$. Then The Bottom-Up quotient for the composition of tree languages is given as follows:

$$\alpha^{-1}(L \circ (L_1, \ldots, L_k)) = \bigcup_{1 \leq j \leq k} L \circ ((\mathrm{Inc}_\varepsilon(1, L_l))_{1 \leq l \leq j}, \alpha^{-1}(L_j), (\mathrm{Inc}_\varepsilon(1, L_l))_{j+1 \leq l \leq k}) \cup \begin{cases} \alpha((\varepsilon_{j_{p_l}})_{1 \leq l \leq n})^{-1}(L) \circ (\varepsilon_1, (\mathrm{Inc}_\varepsilon(1, L_l))_{1 \leq l \leq k | \forall z, l \neq p_z}) \\ \quad \text{if } \forall 1 \leq l \leq n, \exists 1 \leq p_l \leq k, \varepsilon_l \in L_{p_l} \\ \emptyset \quad \text{otherwise.} \end{cases} \tag{2.10}$$

where L is a k-homogeneous language with $\mathrm{Ind}_\varepsilon(L) = \{j_1, \ldots, j_k\}$, $L_1, \ldots, L_k$ are k tree languages.

2.3 Tree Automata

As we have seen in the previous section, the set of tree languages might be infinite, that is why researchers found a way to present this infinite set by a finite object called: *a finite state machine* named a *tree automaton*. This machine is introduced by Thatcher [61] as a generalization of word automata.

The aim of these machines is to read (*recognize*) set of words/trees. Notice that a word over a finite set of symbols is a unary tree (a tree where each symbol is of rank 1 or 0). This structure has two same directions to read it by the automaton (from right to left or from left to right). However, the structure of trees allows the existence of two kind of tree automata depending on the direction of reading the tree (either from the root to the leaves (Top-Down) or from leaves to the root (Bottom-Up)).

As in the case of words, the tree automata can be presented graphically: its states are presented by nodes, initial state are distinguished by an incoming arrow and final states by a double circle, there exist an edge between two nodes if and only if a transition exists between states.

2.3.1 Bottom-Up Tree Automata

In this part, we introduce one kind of tree automata (the *Bottom-Up* tree automaton). The trees of this automaton are interpreted from leaves to root.

Definition 2.3.1. *The Bottom-Up tree automaton is the automaton defined by*

$$A = (\Sigma, Q, Q_F, \delta)$$

where

- Σ *a finite set of ranked alphabets,*
- Q *a finite set of states,*
- $Q_F \subseteq Q$ *the set of initial states,*
- $\delta \subset \bigcup_{k \geq 0}(Q^k \times \Sigma_k \times Q)$ *the set of transition rules.*

The set of transitions 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 extended as the function from $(2^Q)^k \times \Sigma_k$ to 2^Q defined by

$$\delta(Q_1, \ldots, Q_k, f) = \bigcup_{(q_1, \ldots, q_n) \in Q_1 \times \cdots \times Q_k} \delta(q_1, \ldots, q_k, f). \tag{2.11}$$

Finally, we consider the function Δ *from* T_Σ *to* 2^Q *defined by*

$$\Delta(t) = \begin{cases} \delta(a) & \text{if } t = a, \\ \delta(\Delta(t_1), \ldots, \Delta(t_n), f) & \text{if } t = f(t_1, \ldots, t_n). \end{cases} \tag{2.12}$$

Using these definitions, the language $L(A)$ (which represents the set of all accepted trees by the automaton) recognized by the automaton A is the language

$$L(A) = \{t \in T_\Sigma \mid \Delta(t) \cap Q_F \neq \emptyset\}.$$

As in the case of words, a tree automaton might be deterministic (for each state, there exists only one following state specifying by a transition) or non deterministic (for a given state, there can be zero, one or more transitions).

Definition 2.3.2. *A Bottom-Up tree automaton* $A = (\Sigma, Q, Q_F, \delta)$ *is* deterministic *if for any symbol* f *in* Σ_k*, for any* k *states* $q_1, \ldots, q_k$*,* $|\delta(q_1, \ldots, q_k, f)| \leq 1$*. Otherwise, the tree automaton is* non deterministic.

Example 2.3.1. *Let us consider the Bottom-Up tree automata* $A = (\Sigma, Q, Q_F, \delta)$ *and* $A' = (\Sigma, Q, Q_F, \delta')$ *where*

$$\Sigma = \{a, b, g^1, f^2\}, \qquad Q = \{q_1, q_2, q\}, \quad Q_F = \{q\},$$
$$\delta = \{(a, q_1),\ (b, q_2),\ (q, g, q),\ ((q_1, q_2), f, q)\},$$
$$\delta' = \{(a, q_1),\ (b, q_2),\ (q, g, q),\ ((q_1, q_2), f, q),\ (q, g, q_2)\}.$$

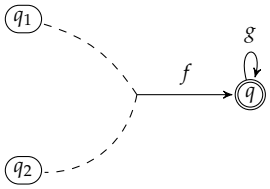


FIGURE 2.3: Deterministic Tree Automaton A.

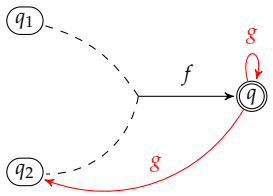
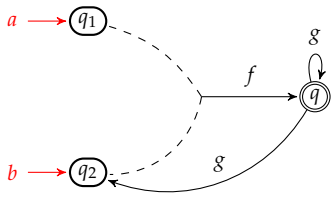


FIGURE 2.4: Non Deterministic Tree Automaton A'.

The automaton A' *is non deterministic because of the existence of the two transitions colored on red in Figure 2.4.*

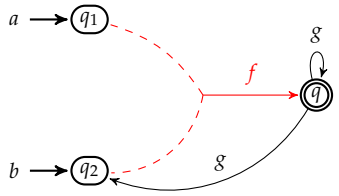
Let us see the behavior of a tree automaton: as we have mentioned before, one of the usefulness of tree automaton is to read\accept\recognize *or to solve* the membership test *of tree languages. Let us verify whether* $t = g(f(a, b))$ *belongs to the language of the automaton* A' *or not. By Definition of* Δ *in Equation (2.12),*

$$\Delta(a) = \delta'(a) = \{q_1\}, \qquad \Delta(b) = \delta'(b) = \{q_2\}.$$



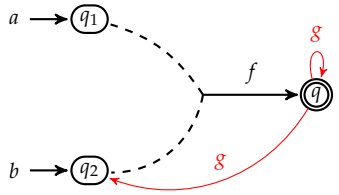
The only transition in δ' labeled by f containing q_a in its first origin and q_2 in its second origin is $((q_1, q_2), f, q)$.

$$\Delta(f(a,b)) = \delta'(\Delta(a), \Delta(b), f) = \{q\}.$$



Last, there are two transitions labeled by g containing q in its origin state and they are: (q, g, q) or (q, g, q_2).

$$\Delta(g(f(a,b)) = \delta'(\Delta(f(a,b), g)) = \{q, q_2\}.$$



Since q is a final state, then $g(f(a,b)) \in L(A')$.

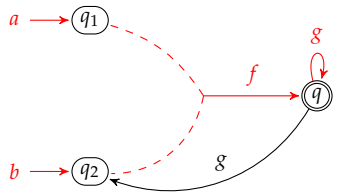


FIGURE 2.5: Recognition of $t = g(f(a,b))$ by the the tree automaton A'.

There exists an algorithm to convert a Bottom-Up non deterministic tree automaton to a Bottom-Up deterministic tree automaton preserving the same language.

Theorem 2.3.1 ([23]). *Let $A' = (\Sigma, Q', Q'_F, \Delta')$ be a non deterministic tree automaton, then there exists a deterministic tree automaton $A = (\Sigma, Q, Q_F, \Delta)$ recognizing the same language.*

2.3.2 Top-Down Tree Automata

In this part, we introduce another kind of tree automata called *Top-Down* tree automaton, the trees of this automaton, as their name mentions, are interpreted from root to leaves.

The Top-Down tree automaton is the automaton defined by

$$A = (\Sigma, Q, Q_I, \delta)$$

where

- Σ a finite set of ranked alphabets,
- Q a finite set of states,
- $Q_I \subseteq Q$ the set of initial states,
- $\delta \subseteq Q \times \Sigma_n \times 2^{Q^n}$ the set of transition rules,

This function can be seen as the function from $2^Q \times \Sigma_n \to Q^n$

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

We consider the function δ^* from $2^Q \times \Sigma_n \to 2^{Q^n}$ defined by

$$\delta^*(Q, f) = \bigcup_{q \in Q} \delta(q, f).$$

Finally, we can consider the function Δ from $2^Q \times \Sigma_n \to 2^{Q^n}$ as follows

$$\Delta(Q, f(t_1, \ldots, t_k)) = \bigcup_{(q_1, \ldots, q_n) \in \delta^*(Q, f)} \Delta(q_1, t_1) \times \ldots \times \Delta(q_n, t_n).$$

The set of trees accepted by the Top-Down tree automaton represents the language $L(A)$ defined by

$$L(A) = \{t \mid \delta(Q_I, t) \neq \emptyset\}.$$

The Top-Down tree automaton is *deterministic* if and only if the set Q_I contains only one initial state and if there exist two transitions $((q_1, \ldots, q_k), f, q),\ ((q'_1, \ldots, q'_k), f, q) \in \delta$ (for $f \in \Sigma_k$, for any state $q, q_1, \ldots, q_k \in Q$) then $(q_1, \ldots, q_k) = (q'_1, \ldots, q'_k)$.

Unlike to the Bottom-Up tree automaton, *determinizing* a non deterministic Top-Down tree automaton does not hold for all instances. As an example, let us consider the non deterministic tree automaton A.

The language denoted by A is $\{f(a, b),\ f(b, a)\}$, let us suppose that the same language is denoted by a deterministic tree automaton A'; in this case, we have one initial state which is q_f. Let us suppose that there exists two states q_a and q_b such that, by Equation (**??**), $\delta(q_f, f(a, b)) = f(\delta(q_a, a), \delta(q_b, b))$ and $\delta(q_f, f(b, a)) = f(\delta(q_a, b), \delta(q_b, a))$, which means that we should have $\delta(q_a, b)$ and $\delta(q_a, a)$. Contradiction, A' is deterministic.

As a conclusion, there exists tree languages that cannot be recognized by a Top-Down deterministic tree automata.

2.4 Regular Tree Expression

In this section, we present another finite object called *a regular tree expression* in order to present an infinite set: the tree languages.

A regular tree expression over Σ can be presented as a tree over $\Sigma \cup \{+,\ \cdot_a,\ {}^{*_a}\}$ where the symbols $+, \cdot_a$ are of arity 2, the ^{*_a} is of arity 1 and a is in Σ_0. As in the case of words, the regular tree expressions are computed inductively.

Definition 2.4.1. A regular tree expression E *over the alphabet* Σ *is inductively defined by:*

$$E = f(E_1, \ldots, E_k), \qquad E = E_1 + E_2,$$
$$E = E_1 \cdot_c E_2, \qquad E = E_1^{*_c},$$

where $k \in \mathbb{N}$, $c \in \Sigma_0$, $f \in \Sigma_k$ *and* $E_1, \ldots, E_k$ *are any* k *regular expressions over* Σ.

A regular tree expression E is *linear*, if each symbol in $\Sigma_{k \geq 1}$ occurs at most once in E. Note that the symbols of rank 0 may appear more than once. We denote by $\overline{E}$ the linearized form of E, which is the expression E where any occurrence of a symbol is indexed by its position in the expression. The set of indexed symbols, called *positions*, is denoted by $\mathrm{Pos}(\overline{E})$. We consider the *delinearization* mapping h sending a linearized expression over its original

unindexed version. As an example, one can consider the delinearization morphism h that sends an indexed alphabet to its unindexed version. Given a language L, we denote by $\phi(L)$ the set $\{\phi(t) \mid t \in L\}$. The *image by* ϕ of an automaton $A = (\Sigma, Q, Q_F, \delta)$ is the automaton $\phi(A) = (\Sigma', Q, F, \delta')$ where

$$\delta' = \{(q_1, \ldots, q_n, \phi(f), q) \mid (q_1, \ldots, q_n, f, q) \in \delta\}.$$

By a trivial induction over the structure of the trees, it can be shown that

$$\phi(L(A)) = L(\phi(A)). \tag{2.13}$$

The aim of regular tree expressions is to recognize a set of tree languages, such language is denoted by $L(E)$ and called *the language denoted by the expression* E.

Definition 2.4.2. *The* language denoted *by* E *is the language* $L(E)$ *inductively defined by*

$$\begin{aligned} L(\emptyset) &= \emptyset, \\ L(f(E_1, \ldots, E_k)) &= \{f(t_1, \ldots, t_k) \mid t_j \in L(E_j), j \leq k\}, \\ L(E_1 + E_2) &= L(E_1) \cup L(E_2), \\ L(E_1 \cdot_c E_2) &= L(E_1) \cdot_c L(E_2), \\ L(E_1^{*_c}) &= L(E_1)^{*_c}, \end{aligned}$$

with $k \in \mathbb{N}$, $c \in \Sigma_0$, $f \in \Sigma_k$ *and* $E_1, \ldots, E_k$ *any* k *regular expressions over* Σ.

Example 2.4.1. *Let us consider the ranked alphabet defined by* $\Sigma = \Sigma_2 \cup \Sigma_1 \cup \Sigma_0$ *where* $\Sigma_2 = \{f\}$, $\Sigma_1 = \{g\}$ *and* $\Sigma_0 = \{a, b\}$. *Let* E *be the tree expression defined by*

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

Hence,

$$\begin{aligned} \overline{E} &= f_1(a, b) \cdot_a g_2(b)^{*_b}, & Pos(\overline{E}) &= \{f_1, g_2, a, b\} \\ L(E) &= L(f(a, b) \cdot_a g(b)^{*_b}) \\ &= L(f(a, b)) \cdot_a L(g(b)^{*_b}) \\ &= L(f(a, b)) \cdot_a L(g(b))^{*_b} \\ &= \{f(a, b)\} \cdot_a \{g(b)\}^{*_b} \\ &= \{f(a, b)\} \cdot_a \{b,\ g(b),\ g(g(b)),\ g(g(g(b))), \ldots\} \\ &= \{f(b, b),\ f(g(b), b),\ f(g(g(b)), b),\ f(g(g(g(b))), b), \ldots\}. \end{aligned}$$

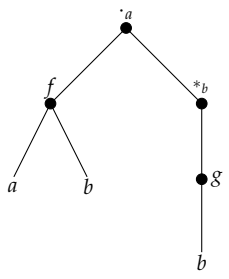


FIGURE 2.6: The syntactic tree of the expression E.

2.5 Conclusion

We have seen throughout this chapter the basic definitions and notations related to tree automata theory. The Figure bellow recapitulates the notions discussed above.

FIGURE 2.7: Recapitulation of Chapter 2.

After defining the structure of trees, the set of tree languages, tree automata and their two interpretations and last regular tree expressions. Let us discover the link between all these mathematical objects in the next chapters.

3 The Constructions of Top-Down Tree Automata

In this chapter, we present conversion methods from a regular tree expression to a Top-Down tree automaton denoting the same language.

In Section 3.1, we show how to convert a regular tree expression into a position tree automaton. We define in Section 3.2 a generalization of *the follow automaton*. Section 3.3 is devoted to present the equation tree automaton. We show in Section 3.4 the construction of the c-continuation tree automaton. Finlay, we summarize in Section 3.5 the link between the introduced automata in the previous sections.

3.1 The Position Tree Automaton

Glushkov [28] and independently Mc-Naughton and Yamada [45] were among the first who could construct a deterministic finite automaton with only $n+1$ states from a regular expression E with n symbols. The basic idea of this construction was to deal with the *linearized* form of a regular expression $\overline{E}$ and to define sets named:

- $\mathrm{Pos}(\overline{E})$ which contains all the indexed symbols of $\overline{E}$.
- $\mathrm{First}(\overline{E})$ which contains the first symbols of the words in the language $L(\overline{E})$.
- $\mathrm{Follow}(\overline{E}, x)$ which is the set containing the successor symbol of the symbol $x \in \overline{E}$ of a word in the language $L(\overline{E})$.
- $Last(\overline{E})$ is the set which contains the symbols that ends the word in the language $L(\overline{E})$.

These sets are used in the computation of the Glushkov automaton or position automaton. This construction runs in time $\mathcal{O}(n^3)$ [28]. Several improvements of this complexity were proposed [11, 52] in order to obtain a quadratic complexity.

The study of Glushkov's method were extended to the computation of tree automata. In this section, we present the results shown in [48, 49] by Mignot *et al*. These latter proposed a constructive algorithm called k-position tree automata (or position tree automata), an extension of Glushkov's one, using a Top-Down interpretation [23]. Such interpretation allows the automaton to start its computation at the root in an initial state and then works down of the tree to the leaves.

3.1.1 Position Functions

Let E be a regular expression over Σ, $\overline{E}$ its linear form and let f be a symbol in Σ_k. Let us define the functions considered in the computation of the tree automaton of this section.

- The set $\mathrm{First}(\overline{E})$, subset of Σ, contains the roots of the trees in the language of the linear expression and is formally defined by

$$\mathrm{First}(\overline{E}) = \{\mathrm{root}(t) \mid t \in L(\overline{E})\} \tag{3.1}$$

- The set $\text{Follow}(\overline{E}, f, k)$, subset of Σ, contains all the successors of f (*i.e.* the symbols that appear directly below it, considering that the tree grows downwards) in the tree t whose f is the root of the k-th subtree of t, *i.e.*

$$\text{Follow}(\overline{E}, f, k) = \{g \in \Sigma \mid \exists t \in L(\overline{E}), \exists s \preceq t, \text{root}(s) = f, \text{k} - \text{child(s)} = g\} \tag{3.2}$$

We can illustrate it in the Figure 3.1 shown bellow.

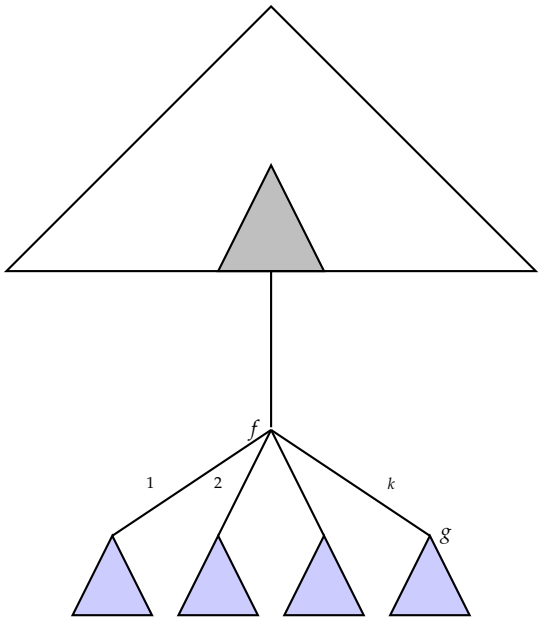


FIGURE 3.1: Illustration of $\text{Follow}(t, f, k)$.

- The set $\text{Last}(\overline{E})$, subset of Σ_0, contains the leaves of the trees in the language of the linear expression and it is defined by

$$\text{Last}(\overline{E}) = \{a \in \Sigma_0 \mid a \preceq t, t \in L(\overline{E})\}. \tag{3.3}$$

As we have seen in Section 2.4, the regular expressions are defined inductively; Therefore, these functions can be computed in the same way over the structure of the linear expression $\overline{E}$:

$$\begin{aligned}
\text{First}(0) = \emptyset, \qquad \text{First}(a) &= \{a\}, \qquad \text{First}(f(\overline{E}_1, ..., \overline{E}_k)) = \{f\}, \\
\text{First}(\overline{E}_1 + \overline{E}_2) &= \text{First}(\overline{E}_1) \cup \text{First}(\overline{E}_2), \\
\text{First}(\overline{E}_1 \cdot_c \overline{E}_2) &= \begin{cases} \text{First}(\overline{E}_1) \setminus \{c\} \cup \text{First}(\overline{E}_2) & \text{if } c \in L(\overline{E}_1), \\ \text{First}(\overline{E}_1) & \text{otherwise,} \end{cases} \\
\text{First}(\overline{E}_1^{*_c}) &= \text{First}(\overline{E}_1) \cup \{c\},
\end{aligned}$$

TABLE 3.1: The computation of the function First.

$$\text{Follow}(0,f,k)=\emptyset, \qquad \text{Follow}(a,f,k)=\emptyset,$$

$$\text{Follow}(g(\overline{E}_1,\ldots,\overline{E}_n),f,k)=\begin{cases}\text{First}(\overline{E}_k) & \text{if } f=g,\\ \text{Follow}(\overline{E}_l,f,k) & \text{if } \exists l \mid f\in\Sigma_{\overline{E}_l},\\ \emptyset & \text{otherwise.}\end{cases}$$

$$\text{Follow}(\overline{E}_1+\overline{E}_2,f,k)=\begin{cases}\text{Follow}(\overline{E}_1,f,k) & \text{if } f\in\Sigma_{\overline{E}_1},\\ \text{Follow}(\overline{E}_2,f,k) & \text{if } f\in\Sigma_{\overline{E}_2},\\ \emptyset & \text{otherwise.}\end{cases}$$

$$\text{Follow}(\overline{E}_1\cdot_c\overline{E}_2,f,k)=\begin{cases}(\text{Follow}(\overline{E}_1,f,k)\setminus\{c\})\cup\text{First}(\overline{E}_2) & \text{if } c\in\text{Follow}(\overline{E}_1,f,k),\\ \text{Follow}(\overline{E}_1,f,k) & \text{if } f\in\Sigma_{\overline{E}_1}\wedge c\notin\text{Follow}(\overline{E}_1,f,k),\\ \text{Follow}(\overline{E}_2,f,k) & \text{if } f\in\Sigma_{\overline{E}_2}\wedge c\in\text{Last}(\overline{E}_1),\\ \emptyset & \text{otherwise ,}\end{cases}$$

$$\text{Follow}(\overline{E}_1^{*_c},f,k)=\begin{cases}\text{Follow}(\overline{E}_1,f,k)\cup\text{First}(\overline{E}_1) & \text{if } c\in\text{Follow}(\overline{E}_1,f,k),\\ \text{Follow}(\overline{E}_1,f,k) & \text{otherwise ,}\end{cases}$$

TABLE 3.2: The computation of the function Follow.

where $\overline{E}_1,\ldots,\overline{E}_k$ are k linear expressions over Σ, f is a symbol in Σ_k, g is a symbol in Σ_n and a,c are symbols in Σ_0.

Example 3.1.1. *Let $\Sigma_0=\{a,b,c\}$, $\Sigma_1=\{g\}$ and $\Sigma_2=\{f\}$. Let us consider the expression over $\Sigma=\Sigma_0\cup\Sigma_1\cup\Sigma_2$ defined by:*

$$E=(a+g(a)+g(a))\cdot_a(f(a,b)^{*_b}\cdot_b g(c)^{*_c})$$

Its linear form is:

$$\overline{E}=(a+g_1(a)+g_2(a))\cdot_a(f_3(a,b)^{*_b}\cdot_b g_4(c)^{*_c})$$

The language denoting the expression $\overline{E}$ is

$$\begin{aligned}L(\overline{E})=\{&c,\ g_1(c),\ g_1(g_4(c)),\ g_1(g_4(g_4(c))),\ g_1(f_3(a,c)),\ g_1(g_4(c)),\ g_1(g_4(g_4(c))),\\ &g_2(c),\ g_2(g_4(g_4(c))),\ g_2(f_3(a,c)),\ f_3(a,c),\ f_3(a,f_3(a,c)),\\ &g_4(c),\ g_4(g_4(c)),\ldots\}\end{aligned}$$

By applying the alphabetical morphism h *we obtain*

$$h(L(\overline{E}))=\{c,\ g(c),\ g(g(c)),\ g(g(g(c))),\ g(f(a,c)),\ f(a,c),\ f(a,f(a,c)),\ldots\}$$

The position functions of the expression $\overline{E}$ are defined by

$$Pos(\overline{E}) = \{g_1, g_2, f_3, g_4, a, b, c\}, \qquad \text{Last}(\overline{E}) = \{a, c\},$$

$$\begin{aligned}
\text{First}(\overline{E}) &= \text{First}((a + g_1(a) + g_2(a)) \cdot_a (f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c})) \\
&= \text{First}(a + g_1(a) + g_2(a)) \setminus \{a\} \cup \text{First}(f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}) \\
&= \text{First}(a) \cup \text{First}(g_1(a)) \cup \text{First}(g_2(a)) \setminus \{a\} \cup \text{First}(f_3(a,b)^{*_b}) \\
&\quad \setminus \{b\} \cup \text{First}(g_4(c)^{*_c}) \\
&= \text{First}(a) \cup \text{First}(g_1(a)) \cup \text{First}(g_2(a)) \setminus \{a\} \cup \text{First}(f_3(a,b)) \\
&\quad \cup \{b\} \setminus \{b\} \cup \text{First}(g_4(c)) \cup \{c\} \\
&= \{a, g_1, g_2, \} \setminus \{a\} \cup \{f_3\} \cup \{b\} \setminus \{b\} \cup \{g_4\} \cup \{c\} \\
&= \{g_1, g_2, f_3, g_4, c\},
\end{aligned}$$

$$\begin{aligned}
\text{Follow}(\overline{E}, g_1, 1) &= \text{Follow}((a + g_1(a) + g_2(a)) \cdot_a (f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}), g_1, 1) \\
&= (\text{Follow}(a + g_1(a) + g_2(a), g_1, 1) \setminus \{a\}) \cup \text{First}(f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}) \\
&= (\text{Follow}(g_1(a) + g_2(a), g_1, 1) \setminus \{a\}) \cup \text{First}(f_3(a,b)^{*_b}) \setminus \{b\} \cup \text{First}(g_4(c)^{*_c}) \\
&= (\text{Follow}(g_1(a), g_1, 1) \setminus \{a\}) \cup \text{First}(f_3(a,b)) \cup \{b\} \setminus \{b\} \cup \text{First}(g_4(c)) \cup \{c\} \\
&= (\{a\} \setminus \{a\}) \cup \{f_3\} \cup \{b\} \setminus \{b\} \cup \{g_4\} \cup \{c\} \\
&= \{f_3, g_4, c\},
\end{aligned}$$

$$\begin{aligned}
\text{Follow}(\overline{E}, g_2, 1) &= \text{Follow}((a + g_1(a) + g_2(a)) \cdot_a (f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}), g_2, 1) \\
&= (\text{Follow}(a + g_1(a) + g_2(a), g_2, 1) \setminus \{a\}) \cup \text{First}(f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}) \\
&= (\text{Follow}(g_1(a) + g_2(a), g_2, 1) \setminus \{a\}) \cup \text{First}(f_3(a,b)^{*_b}) \setminus \{b\} \cup \text{First}(g_4(c)^{*_c}) \\
&= (\text{Follow}(g_2(a), g_2, 1) \setminus \{a\}) \cup \text{First}(f_3(a,b)) \cup \{b\} \setminus \{b\} \cup \text{First}(g_4(c)) \cup \{c\} \\
&= (\{a\} \setminus \{a\}) \cup \{f_3\} \cup \{b\} \setminus \{b\} \cup \{g_4\} \cup \{c\} \\
&= \{f_3, g_4, c\},
\end{aligned}$$

$$\begin{aligned}
\text{Follow}(\overline{E}, f_3, 1) &= \text{Follow}((a + g_1(a) + g_2(a)) \cdot_a (f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}), f_3, 1) \\
&= \text{Follow}(f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}, f_3, 1) \\
&= \text{Follow}(f_3(a,b)^{*_b}, f_3, 1) \\
&= \text{Follow}(f_3(a,b), f_3, 1) \\
&= \{a\},
\end{aligned}$$

$$\begin{aligned}
\text{Follow}(\overline{E}, f_3, 2) &= \text{Follow}((a + g_1(a) + g_2(a)) \cdot_a (f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}), f_3, 2) \\
&= \text{Follow}(f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}, f_3, 2) \\
&= \text{Follow}(f_3(a,b)^{*_b}, f_3, 2) \setminus \{b\} \cup \text{First}(g_4(c)^{*_c}) \\
&= \text{Follow}(f_3(a,b), f_3, 2) \cup \text{First}(f_3(a,b)) \setminus \{b\} \cup \text{First}(g_4(c)) \cup \{c\} \\
&= \{b\} \cup \{f_3\} \setminus \{b\} \cup \{g_4\} \cup \{c\} \\
&= \{f_3, g_4, c\},
\end{aligned}$$

$$\begin{aligned}
\text{Follow}(\overline{E}, g_4, 1) &= \text{Follow}((a + g_1(a) + g_2(a)) \cdot_a (f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}), g_4, 1) \\
&= \text{Follow}(f_3(a,b)^{*_b} \cdot_b g_4(c)^{*_c}, g_4, 1) \\
&= \text{Follow}(g_4(c)^{*_c}, g_4, 1) \\
&= \text{Follow}(g_4(c), g_4, 1) \cup \text{First}(g_4(c)) \\
&= \{c, g_4\}.
\end{aligned}$$

3.1.2 Top-Down Position Tree Automaton

The position functions defined above are used for the computation of the position tree automaton in a Top-Down way which means that the computation of the transitions is done from root to leaves using a function Follow, recognizing the language denoting by the expression E.

The Top-Down position tree automaton $\mathcal{A}_{\mathcal{P}}$ associated to the linear expression $\overline{E}$ is the automaton

$$(Q, \Sigma, \{\varepsilon^1\}, \Delta)$$

where

$$\begin{aligned} Q &= \{f^k \mid f \in \Sigma_m, 1 \leq k \leq m\} \cup \{\varepsilon^1\} \text{ with } \varepsilon^1 \text{ a new symbol not in } \Sigma, \\ \Delta &= \{(f^k, g, (g^1, \ldots, g^n)) \mid g \in \text{Follow}(\overline{E}, f, k)\} \\ &\quad \cup \{(\varepsilon^1, g, (g^1, \ldots, g^n)) \mid g \in \Sigma_n \wedge g \in \text{First}(\overline{E})\} \\ &\quad \cup \{(\varepsilon^1, c) \mid c \in \Sigma_0 \wedge c \in \text{First}(\overline{E})\} \end{aligned}$$

For the purpose of proving that the language of the Top-Down position tree automaton recognizes the language of the regular tree expression $\overline{E}$, the authors of [48, 19] gave a characterization of the membership of a tree $t = f(t_1, \ldots, t_k)$ in the language denoted by $\overline{E}$ as follows:

First of all, they showed that a given tree t is belonging to a language of a regular expression if and only if the set $\text{First}(\overline{E})$ contains the root of t and for all $g(k_1, \ldots, k_l)$, subtree of t, the function $\text{Follow}(\overline{E}, g, k)$ contains the roots of the subtrees t_i for $i \in \{1, \ldots, l\}$.

Furthermore, they showed the connection between this characterization and the transition rules Δ of the Top-Down position tree automaton $\mathcal{A}_{\mathcal{P}}$.

As a result, the Top-Down position tree automaton $\mathcal{A}_{\mathcal{P}}$ recognizes exactly the same language denoting the linear expression $\overline{E}$. Replacing each transition $(f_j^k, g_i, (g_i^1, \ldots, g_i^n))$ of $\mathcal{A}_{\mathcal{P}}$ by $(f_j^k, h(g_i), (g_i^1, \ldots, g_i^n))$ by using the alphabetical morphism h, we get $h(L(\mathcal{A}_{\mathcal{P}})) = L(E)$.

Example 3.1.2. *Let us consider the regular expression defined in Example (3.1.1) by*

$$E = (a + g(a) + g(a)) \cdot_a (f(a, b)^{*_b} \cdot_b g(c)^{*_c})$$

The expression $\overline{E}$ is given by

$$\overline{E} = (a + g_1(a) + g_2(a)) \cdot_a (f_3(a, b)^{*_b} \cdot_b g_4(c)^{*_c})$$

The set of states is

$$Q = \{\varepsilon^1, g_1^1, g_2^1, f_3^1, f_3^2, g_4^1\}$$

The set of transition rules is

$$\begin{aligned} \Delta = \{ & \\ & (\varepsilon^1, c), \quad (\varepsilon^1, g_1, g_1^1), \quad (\varepsilon^1, g_2, g_2^1), \quad (\varepsilon^1, f_3, (f_3^1, f_3^2)), \quad (\varepsilon^1, g_4, g_4^1), \\ & (g_1^1, c), \quad (g_1^1, g_4, g_4^1), \quad (g_1^1, f_3, (f_3^1, f_3^2)), \quad (g_2^1, c), \quad (g_2^1, g_4, , g_4^1), \\ & (g_2^1, f_3, (f_3^1, f_3^2)), \quad (f_3^1, c), \quad (f_3^2, c), \quad (f_3^2, f_3, (f_3^1, f_3^2)), \quad (f_3^2, g_4, g_4^1), \\ & (g_4^1, c), \quad (g_4^1, g_4, g_4^1) \\ & \} \end{aligned}$$

The Top-Down position tree automaton associated to $\overline{E}$ is shown in Figure 3.2.

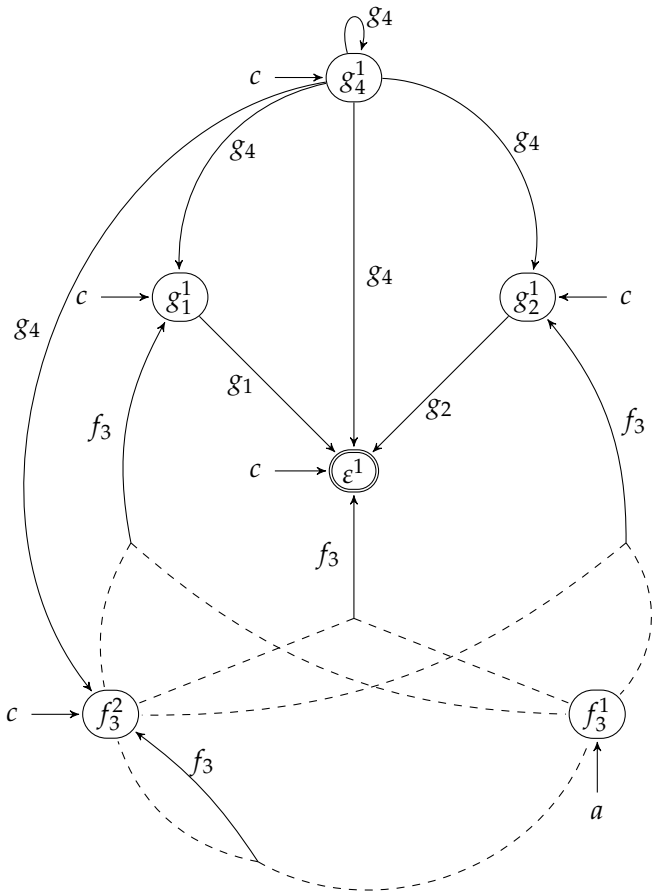


FIGURE 3.2: The Top-Down Position Tree Automaton of $\overline{E}$.

3.2 The Follow Tree Automaton

Ilie and Yu constructed a non deterministic finite automaton from a regular expression denoted the same language, named *follow automata* [35]. This method is an improvement of the Glushkov's algorithm and it is computed in a quadratic time [35]. Its basic idea is to identify the states of the position automaton having the same Follow set and then merge them.

In this section, we give a computation of an extension of [35] in the case of words to tree automata called *follow tree automaton*. This computation is given in [48, 49]. The states of this automaton are sets of positions instead of positions in the case of position tree automaton already defined in Section 3.1 and the function Follow already seen in Section 3.1.1 is extended by $\text{Follow}(E, \varepsilon^1, 1) = \text{First}(E)$. Moreover, we show that the follow tree automaton is the *quotient* of position tree automaton in the similar way as word automata proven by Ilie and Yu [35]. The results of this section are extracted from [48, 49].

The follow tree automaton $\mathcal{A}_{\mathcal{F}}$ associated with the linear expression $\overline{E}$ is the automaton

$$(Q, \Sigma, Q_T, \Delta)$$

where

$$\begin{aligned} Q &= \{\text{First}(\overline{E})\} \cup \{ \bigcup_{f \in \Sigma_m} \{\text{Follow}(\overline{E}, f, k) \mid 1 \leq k \leq m\} \\ Q_T &= \{\text{First}(\overline{E})\} \\ \Delta &= \{(\text{Follow}(\overline{E}, g, l),\ f,\ \text{Follow}(\overline{E}, f, 1), \ldots, \text{Follow}(\overline{E}, f, m)) \mid f \in \Sigma_m \\ &\quad \wedge f \in \text{Follow}(\overline{E}, g, l) \wedge g \in \Sigma_n \cup \{\varepsilon^1\} \wedge l \leq n\} \\ &\quad \cup \{(I, c) \mid I \in Q, c \in I \wedge c \in \Sigma_0\} \end{aligned}$$

In order to show that the follow automaton recognizes the same language as the regular expression, let us first see that the follow tree automaton is the quotient of the Top-Down position tree automaton considering the fact that this quotient conserves the language.

Given $\overline{E}$, a regular tree expression over Σ, and considering $\mathcal{A}_{\mathcal{P}} = (Q, \Sigma, \{\varepsilon\}, \Delta)$ the Top-Down position tree automaton of $\overline{E}$. Quotienting $\mathcal{A}_{\mathcal{P}}$ to $\mathcal{A}_{\mathcal{F}}$ is given by *the follow relation* $\sim_{\mathcal{F}}$ in Q which is defined by

$$f^k \sim_{\mathcal{F}} g^l \Leftrightarrow \text{Follow}(\overline{E}, f, k) = \text{Follow}(\overline{E}, g, l).$$

where f^k, g^l are two states of Q.

This relation allows us to compute the follow tree automaton from the Top-Down position tree automaton by identifying the similar states of $\mathcal{A}_{\mathcal{P}}$ and then to merge them into one state.

We have seen in Section 6.1 that the language recognized by a linear expression is the same denoted by $\mathcal{A}_{\mathcal{P}}$. Furthermore, since $\mathcal{A}_{\mathcal{F}}$ is the quotient of $\mathcal{A}_{\mathcal{P}}$ w.r.t the follow relation, $L(\mathcal{A}_{\mathcal{P}}) = L(\text{A}_\text{F})$. Therefore, the language denoted by $\overline{E}$ is recognized by the follow tree automaton too.

Finally, we apply the alphabetical morphism h by replacing each transition $(\text{Follow}(\overline{E}, g, l),$ $f_j,$ $\text{Follow}(\overline{E}, f, 1), \ldots, \text{Follow}(\overline{E}, f, m))$ of $\mathcal{A}_{\mathcal{F}}$ by $(\text{Follow}(E, g, l),$ $h(f_j),$ $\text{Follow}(E, f, 1),$ $\ldots, \text{Follow}(E, f, m))$, we obtain $h(L(\mathcal{A}_{\mathcal{F}})) = L(E)$.

Let us see an example of how to construct the follow tree automaton from the position tree automaton.

Example 3.2.1. *Let us consider the constructed position tree automaton of Example 3.1.2.*

Using the equivalence relation $\sim_F$ over the set of states of the position tree automaton $\mathcal{A}_{\mathcal{P}}$.

$$\text{Follow}(\overline{E}, g_1, 1) = \text{Follow}(\overline{E}, g_2, 1) = \text{Follow}(\overline{E}, f_3, 2)$$

$$\Downarrow$$

$$g_1^1 \sim_{\mathcal{F}} g_2^1 \sim_{\mathcal{F}} f_3^2.$$

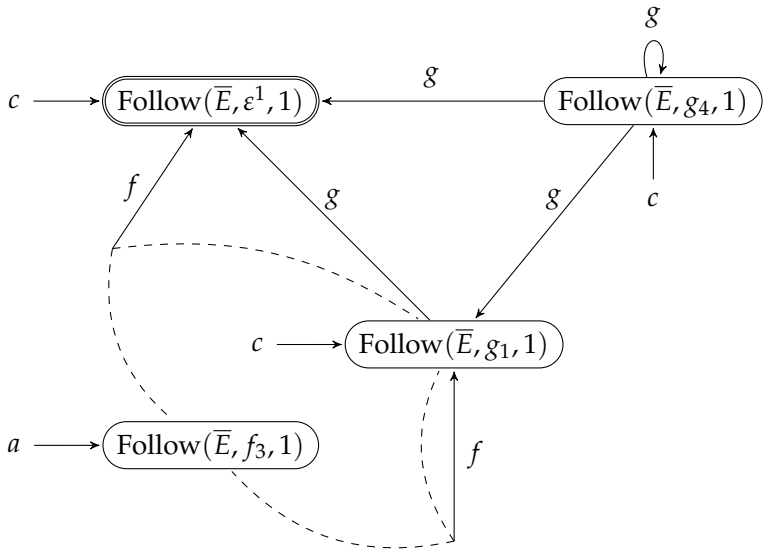


FIGURE 3.3: The Automaton $\mathcal{A}_{\mathcal{P}}/\sim_F$.

Let us compute the follow tree automaton $\mathcal{A}_{\mathcal{F}}$ associated to $\overline{E}$.

Example 3.2.2. *Let us consider the tree expression* $E = (a + g(a) + g(a)) \cdot_a (f(a,b)^{*_b} \cdot_b g(c)^{*_c})$ *defined in Example 3.1.1 and* $\overline{E}$ *be its linear form.*
The set of states is

$$\begin{aligned} Q &= \text{First}(\overline{E}) \cup \text{Follow}(\overline{E}, g_1, 1) \cup \text{Follow}(\overline{E}, g_2, 1) \cup \text{Follow}(\overline{E}, f_3, 1) \cup \text{Follow}(\overline{E}, f_3, 2) \\ &\quad \cup \text{Follow}(\overline{E}, g_4, 1) \\ &= \{\{c, g_1, g_2, f_3, g_4\}, \{c, f_3, g_4\}, \{c, f_3, g_4\}, \{a\}, \{c, f_3, g_4\}, \{c, g_4\}\} \\ &= \{\{c, g_1, g_2, f_3, g_4\}, \{c, f_3, g_4\}, \{a\}, \{c, f_3, g_4\}, \{c, g_4\}\}. \end{aligned}$$

The set of transition rules is

$$\begin{aligned} \Delta = \{ & \\ & (\{c, g_1, g_2, f_3, g_4\}, g_1, \{c, f_3, g_4\}), && (\{c, g_1, g_2, f_3, g_4\}, g_2, \{c, f_3, g_4\}), \\ & (\{c, g_1, g_2, f_3, g_4\}, f_3, \{a\}, \{c, f_3, g_4\}), && (\{c, f_3, g_4\}, f_3, \{a\}, \{c, f_3, g_4\}), \\ & (\{c, g_1, g_2, f_3, g_4\}, g_4, \{c, g_4\}), && (\{c, f_3, g_4\}, g_4, \{c, g_4\}), \\ & (\{c, g_4\}, g_4, \{c, g_4\}), && (\{c, g_1, g_2, f_3, g_4\}, c), \\ & (\{a\}, a), \qquad (\{c, f_3, g_4\}, c), && (\{c, g_4\}, c) \\ \} & \end{aligned}$$

The follow tree automaton is presented in Figure 3.4

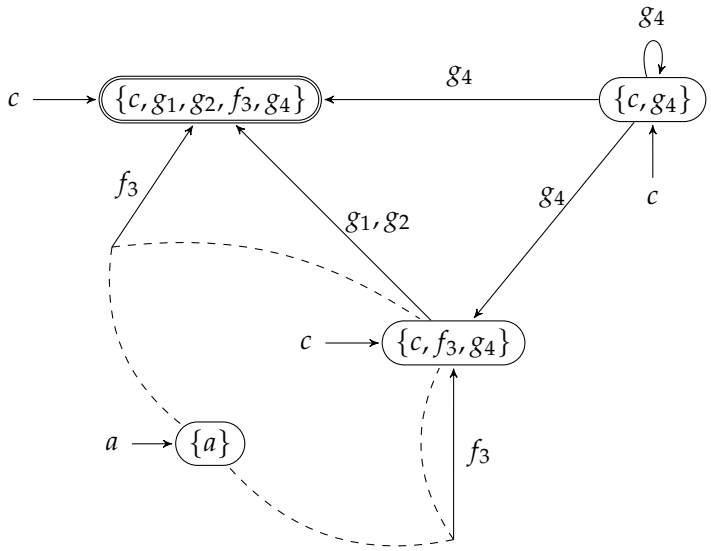


FIGURE 3.4: The Follow Tree Automaton of $\overline{E}$.

3.3 The Equation Tree Automaton

Brzozowski [13] gave an algorithm to convert a regular expression into a deterministic finite automaton recognizing the same language. The idea of this conversion is based on the notion of language quotient over a regular expression by a word w *i.e.* $w^{-1}(L(E))$ and on an operation denoted by $d_w(E)$ (the derivative of the expression E w.r.t a word w). This latter represents a regular expression denoting the set containing the words w' such that $ww' \in L(E)$ and they constitute the states of the derivative automaton. Brzozowski [13] showed that computing the language quotient denoted by a regular expression w.r.t a word w labeled in the path of a given state to a final state, is the same as computing the language denoting the derived regular expression w.r.t w.

The obtained automaton has a finite number of states only by considering three rules: Associativity, Commutativity and Idempotent of the sum. However, even if the number of states is finite, it is exponential. That is why in 1996 Antimirov [2] tried to fix the "limit" of this automaton by suggesting another construction based on *partial derivative* denoted by $\partial_a(E)$ (the partial derivative of E w.r.t a). Its main idea is to replace an expression by a set of expression. The constructed automaton is non deterministic and has at most $n + 1$ states where n is the number of symbols of E.

The study of this construction is extended to the notion of trees. This section is devoted to recall another construction of tree automaton from a regular tree expression called *equation tree automaton* due to Kuske and Meinecke [42]. The principal idea of this construction is to extend the notion of *partial derivatives* due to Antimirov [2] to tree partial derivative where the result of such derivative is sets of tuples of expressions instead of sets of regular expression. The content of this section contains the results of [42] using the notations of [49].

Let E be a regular tree expression, let f be a symbol in Σ_k and c be a symbol in Σ_0. Let $f^{-1}(E)$ be the set of tuples of regular expression and it is inductively defined by

$$\begin{aligned}
f^{-1}(0) &= \emptyset, \\
f^{-1}(E_1 + E_2) &= f^{-1}(E_1) \cup f^{-1}(E_2), \\
f^{-1}(g(E_1, \ldots, E_k)) &= \begin{cases} \{(E_1, \ldots, E_k)\} & \text{if } f = g, \\ \emptyset & \text{otherwise,} \end{cases} \\
f^{-1}(E_1 \cdot_c E_2) &= \begin{cases} f^{-1}(E_1) \cdot_c E_2 & \text{if } c \notin L(E_1), \\ f^{-1}(E_1) \cdot_c E_2 \cup f^{-1}(E_2) & \text{otherwise,} \end{cases} \\
f^{-1}(E_1^{*_c}) &= f^{-1}(E_1) \cdot_c E_1^{*_c},
\end{aligned}$$

where $(E_1, \ldots, E_k)$ are k tree expressions and where the derivative formulas of the c-product and the c-closure is the c-product of a tuple of expressions and a regular expression defined by

$$\mathcal{E} \cdot_c E = (E_1 \cdot_c E, \ldots, E_k \cdot_c E)$$

where $\mathcal{E} = (E_1, \ldots, E_k)$.

Moreover, the derivative formulas f^{-1} is also extended to any set of tuples of regular expression. Thus, by considering $\mathcal{S}$ a set of tuples of regular expression,

$$f^{-1}(\mathrm{S}) = \bigcup_{E \in \mathcal{S}} f^{-1}(E)$$

Finally, we denote by $\partial_w(E)$ the partial derivative of a regular expression E w.r.t a word $w \in \Sigma_{\geq 1}^*$ and it is the set of regular expression defined by

$$\partial_w(E) = \begin{cases} \{E\} & \text{if } w = \varepsilon, \\ \mathrm{SET}(f^{-1}(\partial_u(E))) & \text{if } w = uf \wedge u \in \Sigma_{\geq 1}^* \wedge f^{-1}(\partial_u(E)) \neq \emptyset, \\ \{0\} & \text{if } w = uf \wedge u \in \Sigma_{\geq 1}^* \wedge f^{-1}(\partial_u(E)) = \emptyset. \end{cases}$$

where $\mathrm{SET}(E_1, \ldots, E_n) = \{E_1, \ldots, E_n\}$.

The result of a partial derivative of a regular expression w.r.t $w \in \Sigma_{\geq 1}^*$ is a set of expression called *derived term* of E.

Example 3.3.1. *Let us consider the alphabet defined by* $\Sigma_2 = \{f\}$, $\Sigma_1 = \{g\}$, $\Sigma_0 = \{a, b, c\}$ *and* $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$. *Let* E *be the tree expression defined by* $E = E_1 \cdot_a E_2 \cdot_b E_3$.

where $E_1 = (a + g(a) + g(a))$, $E_2 = f(a, b)^{*_b}$ *and* $E_3 = g(c)^{*_c}$.

Let us show how to calculate the derivative of E *w.r.t* $w \in \Sigma_{\geq 1}^*$.

$$\begin{aligned}\partial_f(E) &= \partial_f(E_1 \cdot_a E_2 \cdot_b E_3)\\ &= \partial_f(E_1 \cdot_a E_2) \cdot_b E_3 \cup \partial_f(E_3)\\ &= (\partial_f(E_1) \cdot_a E_2 \cup \partial_f(E_2)) \cdot_b E_3 \cup \partial_f(E_3)\\ &= \partial_f(E_2) \cdot_b E_3\\ &= \{a \cdot_b f(a,b)^{*_b} \cdot_b E_3,\ \ b \cdot_b f(a,b)^{*_b} \cdot_b E_3\}\\ &= \{a \cdot_b E_2 \cdot_b E_3,\ b \cdot_b E_2 \cdot_b E_3\}\end{aligned}$$

$$\begin{aligned}\partial_g(E) &= \partial_g(E_1 \cdot_a E_2 \cdot_b E_3)\\ &= (\partial_g(E_1) \cdot_a E_2) \cdot_b E_3 \cup \partial_g(E_3)\\ &= (\partial_g(E_1) \cdot_a E_2 \cup \partial_g(E_2)) \cdot_b E_3 \cup \partial_g(E_3)\\ &= \{a \cdot_a E_2 \cdot_b E_3\} \cup \{c \cdot_c g(c)^{*_c}\}\\ &= \{a \cdot_a E_2 \cdot_b E_3,\ \ c \cdot_c E_3\}\end{aligned}$$

$$\begin{aligned}\partial_{ff}(E) &= \partial_{ff}(E_1 \cdot_a E_2 \cdot_b E_3)\\ &= \partial_f(\partial_f(E_1 \cdot_a E_2 \cdot_b E_3)\\ &= \partial_f(a \cdot_b E_2 \cdot_b E_3) \cup \partial_f(b \cdot_b E_2 \cdot_b E_3)\\ &= \partial_f(E_2 \cdot_b E_3)\\ &= \partial_f(E_2) \cdot_b E_3 \cup \partial_f(E_3)\\ &= \{a \cdot_b f(a,b)^{*_b} \cdot_b E_3,\ b \cdot_b f(a,b)^{*_b}, \cdot_b E_3\}\\ &= \{a \cdot_b E_2 \cdot_b E_3,\ b \cdot_b E_2 \cdot_b E_3\}\\ &= \partial_f(E)\end{aligned}$$

$$\begin{aligned}\partial_{gf}(E) &= \partial_{gf}(E_1 \cdot_a E_2 \cdot_b E_3)\\ &= \partial_g(\partial_f(E_1 \cdot_a E_2 \cdot_b E_3))\\ &= \partial_g(a \cdot_b E_2 \cdot_b E_3) \cup \partial_g(b \cdot_b E_2 \cdot_b E_3)\\ &= \partial_g(b \cdot_b E_3 \cdot_b E_4)\\ &= \partial_g(E_2 \cdot_b E_3)\\ &= \partial_g(E_2) \cdot_b E_3 \cup \partial_g(E_3)\\ &= \partial_g(g(c)^{*_c})\\ &= \{c \cdot_c g(c)^{*_c}\}\\ &= \{c \cdot_c E_3\}\end{aligned}$$

$$\begin{aligned}\partial_{fg}(E) &= \partial_{fg}(E_1 \cdot_a E_2 \cdot_b E_3)\\ &= \partial_f(\partial_g(E_1 \cdot_a E_2 \cdot_b E_3))\\ &= \partial_f(a \cdot_a E_2 \cdot_b E_3) \cup \partial_f(c \cdot_c E_3)\\ &= \partial_f(E_2 \cdot_b E_3)\\ &= \partial_f(E_2) \cdot_b E_3 \cup \partial_f(E_3)\\ &= \partial_f(E_2) \cdot_b E_3\\ &= \{a \cdot_b E_2 \cdot_b E_3,\ b \cdot_b E_2 \cdot_b E_3\}\\ &= \partial_f(E)\end{aligned}$$

$$\begin{aligned}\partial_{gg}(E) &= \partial_{gg}(E_1 \cdot_a E_2 \cdot_b E_3)\\ &= \partial_g(\partial_g(E_1 \cdot_a E_2 \cdot_b E_3))\\ &= \partial_g(a \cdot_a E_2 \cdot_b E_3) \cup \partial_g(c \cdot_c E_3)\\ &= \partial_g(E_2 \cdot_b E_3) \cup \partial_g(E_3)\\ &= (\partial_g(E_2) \cdot_b E_3 \cup \partial_g(E_3)) \cup \partial_g(E_3)\\ &= \partial_g(E_3)\\ &= \partial_g(c \cdot_c g(c)^{*_c})\\ &= \{c \cdot_c E_3\}.\\ &= \partial_{gf}(E)\end{aligned}$$

$$\partial_{gfg}(E) = \partial_{ggg}(E) = \{c \cdot_c E_3\}$$

Notice that the set of partial derivatives of E, denoted by $\mathcal{D}_E$, w.r.t $w \in \Sigma^*_{\geq 1}$ is finite an it is given by

$$\mathcal{D}_E = \{E,\ a \cdot_b E_2 \cdot_b E_3,\ \ b \cdot_b E_2 \cdot_b E_3,\ \ c \cdot_c E_3,\ \ a \cdot_a E_2 \cdot_b E_3\}.$$

After defining the partial derivative of the expression E, we can define the equation tree automaton

$$\mathcal{A}_{\mathcal{E}} = (Q, \Sigma, \{E\}, \Delta)$$

where

$$\begin{aligned}Q &= \mathcal{D}_E\\ \Delta &= \{(F, f, G, \ldots, G_k) \mid F \in Q, f \in \Sigma_k, k \geq, (G_1, \ldots, G_k) \in f^{-1}(F)\}\\ &\quad \cup \{(F, c) \mid F \in Q \wedge c \in (L(F) \cap \Sigma^0)\}\end{aligned}$$

This tree automaton and the regular tree expression associated with it denote the same language.

Example 3.3.2. *Let us consider the tree expression* $E = E_1 \cdot_a E_2 \cdot_b E_3$.
where $E_1 = (a + g(a) + g(a))$, $E_2 = f(a, b)^{*_b}$ *and* $E_3 = g(c)^{*_c}$ *defined in Example 3.1.1. Let us compute the equation tree automaton* $\mathcal{A}_\mathcal{E}$ *associated to* E.

$$
\begin{aligned}
Q &= \mathcal{D}_E \\
&= \{E,\ \ a \cdot_b E_2 \cdot_b E_3,\ \ b \cdot_b E_2 \cdot_b E_3,\ \ c \cdot_c E_3,\ \ a \cdot_a E_2 \cdot_b E_3\}
\end{aligned}
$$

$$
\begin{aligned}
\Delta = \{ & \\
& (E,\ f,\ a \cdot_b E_2 \cdot_b E_3,\ b \cdot_b E_2 \cdot_b E_3), \qquad (a \cdot_a E_2 \cdot_b E_3,\ f,\ a \cdot_b E_2 \cdot_b E_3,\ b \cdot_b E_2 \cdot_b E_3), \\
& (b \cdot_b E_2 \cdot_b E_3,\ f,\ a \cdot_b E_2 \cdot_b E_3,\ b \cdot_b E_2 \cdot_b E_3), \qquad (b \cdot_b E_2 \cdot_b E_3,\ g,\ c \cdot_c E_3), \\
& (E,\ g,\ a \cdot_a E_2 \cdot_b E_3), \qquad (a \cdot_a E_2 \cdot_b E_3,\ g,\ c \cdot_c E_3), \qquad (E,\ g,\ c \cdot_c E_3), \\
& (E,\ c), \qquad (a \cdot_a E_2 \cdot_b E_3,\ c), \qquad (a \cdot_b E_2 \cdot_b E_3,\ a), \qquad (c \cdot_c E_3,\ c) \\
& (b \cdot_b E_2 \cdot_b E_3,\ c) \\
& \}
\end{aligned}
$$

The equation tree automaton associated to the expression E *is shown in the Figure 3.5.*

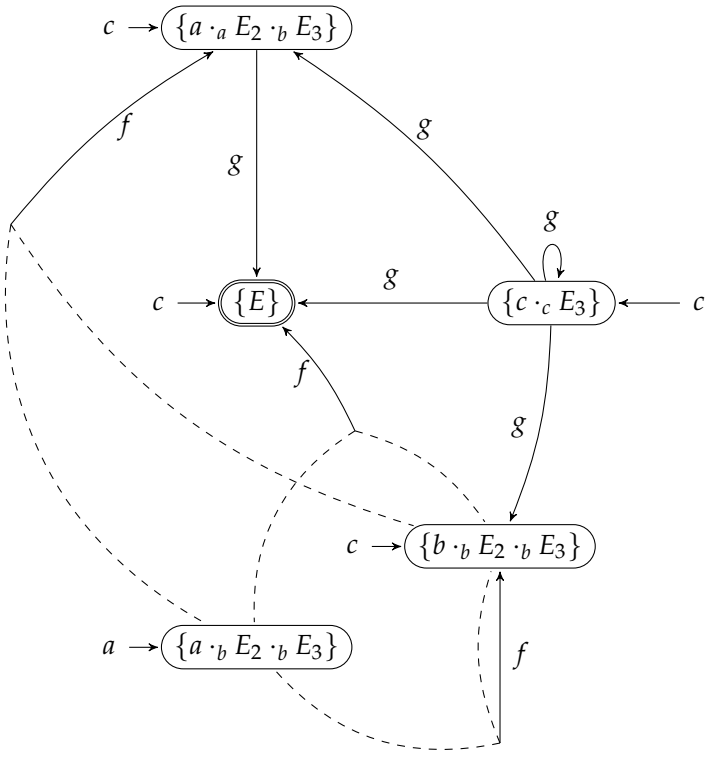


FIGURE 3.5: The Equation Tree Automaton of E.

3.4 The C-Continuation Tree Automaton

This section is dedicated to recall the last Top-Down construction of tree automata of this chapter, called the *c-continuation* tree automaton presented in [47]. In the case of words, this construction were first introduced by Berry and Sethi [9] and extended by Champarnaud

and Ziadi [18, 17]. The idea of this method is to apply the notion of derivation due to Brzozowski [13], to a linear form of a regular expression instead to a regular expression.

Moreover, Champarnaud and Ziadi proved in [17] that the position automata is *isomorphic* to the c-continuation automaton in the case of words. In the similar way, it is shown that this relation still valid in the case of trees too [48, 49].

The content of this section is extracted from [47, 48, 49] and we only consider regular expressions without 0 or reduced to 0 (*i.e.* without occurrences of 0).

We denote by $C_{f^k}(\overline{E})$ the c-continuation of f in $\overline{E}$ and it is the expression defined by

$$C_{f^k}(g(\overline{E_1},\ldots,\overline{E_m})) = \begin{cases} \overline{E}_k & \text{if } f = g \\ C_{f^k}(\overline{E}_j) & \text{if } f \in \Sigma_{\overline{E}_j} \end{cases}$$

$$C_{f^k}(\overline{E}_1 + \overline{E}_2) = \begin{cases} C_{f^k}(\overline{E}_1) & \text{if } f \in \Sigma_{\overline{E}_1} \\ C_{f^k}(\overline{E}_2) & \text{if } f \in \Sigma_{\overline{E}_2} \end{cases}$$

$$C_{f^k}(\overline{E}_1 \cdot_c \overline{E}_2) = \begin{cases} C_{f^k}(\overline{E}_1) \cdot_c \overline{E}_2 & \text{if } f \in \Sigma_{\overline{E}_1} \\ C_{f^k}(\overline{E}_1) & \text{if } f \in \Sigma_{\overline{E}_2} \wedge c \in \mathrm{Last}(\overline{E}_1) \\ 0 & \text{otherwise} \end{cases}$$

$$C_{f^k}(\overline{E}_1^{*_c}) = C_{f^k}(\overline{E}_1) \cdot_c \overline{E}_1^{*_c}$$

where $\overline{E} \neq 0$ is a linear form of a regular expression E, k and m are two integers where $1 \leq k \leq m$ and f is a symbol in $\Sigma_{\overline{E}} \cap \Sigma_m$.

We set by convention $C_\varepsilon(E) = E$.

Example 3.4.1. *Let us consider the tree expression* $E = (a + g(a) + g(a)) \cdot_a (f(a,b)^{*_b} \cdot_b g(c)^{*_c})$ *defined in Example 3.1.1 and* $\overline{E} = E_1 \cdot_a E_2 \cdot_b E_3$ *be its linear form. Where* $E_1 = (a + g_1(a) + g_2(a))$, $E_2 = f_3(a,b)^{*_b}$ *and* $E_3 = g_4(c)^{*_c}$.

Let us compute the c-continuation of $\overline{E}$.

$$\begin{aligned} C_{g_1^1}(\overline{E}) &= C_{g^1}(E_1 \cdot_a E_2 \cdot_b E_3) \\ &= C_{g_1^1}(E_1) \cdot_a (E_2 \cdot_b E_3) \\ &= C_{g_1^1}(a + g_1(a) + g_2(a)) \cdot_a (E_2 \cdot_b E_3) \\ &= a \cdot_a (E_2 \cdot_b E_3) \end{aligned}$$

$$\begin{aligned} C_{g_2^1}(\overline{E}) &= C_{g_2^1}(E_1 \cdot_a E_2 \cdot_b E_3) \\ &= C_{g_2^1}(E_1) \cdot_a (E_2 \cdot_b E_3) \\ &= a \cdot_a (E_2 \cdot_b E_3) \end{aligned}$$

$$\begin{aligned} C_{f_3^1}(\overline{E}) &= C_{f_3^1}(E_1 \cdot_a E_2 \cdot_b E_3) \\ &= C_{f_3^1}(E_2 \cdot_b E_3) \\ &= C_{f_3^1}(E_2) \cdot_b E_3 \\ &= C_{f_3^1}(f_3(a,b)^{*_b}) \cdot_b E_3 \\ &= C_{f_3^1}(f_3(a,b)) \cdot_b f_3(a,b)^{*_b} \cdot_b E_3 \\ &= a \cdot_b E_2 \cdot_b E_3 \end{aligned}$$

$$\begin{aligned} C_{f_3^2}(\overline{E}) &= C_{f_3^2}(E_1 \cdot_a E_2 \cdot_b E_3) \\ &= C_{f_3^2}(E_2 \cdot_b E_3) \\ &= C_{f_3^2}(E_2) \cdot_b E_3 \\ &= C_{f_3^2}(f_3(a,b)^{*_b}) \cdot_b E_3 \\ &= C_{f_3^2}(f_3(a,b)) \cdot_b f_3(a,b)^{*_b} \cdot_b E_3 \\ &= b \cdot_b E_2 \cdot_b E_3 \end{aligned}$$

$$\begin{aligned} C_{g_4^1}(\overline{E}) &= C_{g_4^1}(E_1 \cdot_a E_2 \cdot_b E_3) \\ &= C_{g_4^1}(E_2 \cdot_b E_3) \\ &= C_{g_4^1}(E_3) \\ &= C_{g_4^1}(g_4(c)^{*_c}) \\ &= C_{g_4^1}(g_4(c)) \cdot_c g_4(c)^{*_c} \\ &= c \cdot_c E_3 \end{aligned}$$

After defining the $C_{f^k}(\overline{E})$, now we are able to compute from the regular expression $\overline{E}$ the c-continuation tree automaton

$$\mathcal{A}_{\mathcal{C}} = (Q_{\mathcal{C}}, \Sigma_{\overline{E}}, \{C_{\varepsilon^1}(\overline{E})\}, \Delta_{\mathcal{C}})$$

where

$$\begin{aligned} Q_{\mathcal{C}} &= \{(f^k, C_{f^k}(\overline{E})) \mid f \in \Sigma_m, 1 \leq k \leq m\} \cup \{(\varepsilon^1, C_{\varepsilon^1}(\overline{E})\} \\ \Delta_{\mathcal{C}} &= \{(x, C_x(\overline{E})), g, ((g^1, C_{g^1}(\overline{E})), \ldots, (g^m, C_{g^m}(\overline{E})))) \\ &\quad \mid g \in \Sigma_{\overline{E}}, m \geq 1, (C_{g^1}(\overline{E}), \ldots, C_{g^m}(\overline{E})) \in g^{-1}(C_x(\overline{E}))\} \\ &\quad \cup \{((x, C_x(\overline{E})), c) \mid c \in L(C_x(\overline{E})) \cap \Sigma_0\} \end{aligned}$$

Using the mapping h by unmarking the transitions of $\mathcal{A}_{\mathcal{C}}$ we obtain the c-continuation tree automaton associated to E. This constructed automaton recognizes the language denoting the regular expression [47, 48].

Example 3.4.2. *Let us construct the c-continuation tree automaton from a regular tree expression defined in Example 3.1.1 using the computation of the c-continuation of Example 3.4.1.*
The set of states is

$$\begin{aligned} Q = \{&(\varepsilon^1, E_1 \cdot_a E_2 \cdot_b E_3),\ (g_1^1, a \cdot_a E_2 \cdot_b E_3),\ (g_2^1, a \cdot_a E_2 \cdot_b E_3),\ (f_3^1, a \cdot_b E_2 \cdot_b E_3), \\ &(f_3^2, b \cdot_b E_2 \cdot_b E_3),\ (g_4^1, c \cdot_c E_3)\} \end{aligned}$$

The set of transition rules is

$$
\begin{aligned}
\Delta = \{ & \\
& ((\varepsilon^1, E_1 \cdot_a E_2 \cdot_b E_3),\ f_3,\ (f_3^1, a \cdot_b E_2 \cdot_b E_3),\ (f_3^2, b \cdot_b E_2 \cdot_b E_3)), \\
& ((f_3^2, b \cdot_b E_2 \cdot_b E_3),\ f_3,\ (f_3^1, a \cdot_b E_2 \cdot_b E_3),\ (f_3^2, b \cdot_b E_2 \cdot_b E_3)), \\
& ((g_1^1, a \cdot_a E_2 \cdot_b E_3),\ f_3,\ (f_3^1, a \cdot_b E_2 \cdot_b E_3),\ (f_3^2, b \cdot_b E_2 \cdot_b E_3)), \\
& ((\varepsilon^1, E_1 \cdot_a E_2 \cdot_b E_3),\ g_1,\ (g_1^1, a \cdot_a E_2 \cdot_b E_3)), \\
& ((\varepsilon^1, E_1 \cdot_a E_2 \cdot_b E_3),\ g_2,\ (g_2^1, a \cdot_a E_2 \cdot_b E_3)), \\
& ((\varepsilon^1, E_1 \cdot_a E_2 \cdot_b E_3),\ g_4,\ (g_4^1, c \cdot_c E_3)), \\
& ((g_1^1, a \cdot_a E_2 \cdot_b E_3),\ g_4,\ (g_4^1, c \cdot_c E_3)), \\
& ((f_3^2, b \cdot_b E_2 \cdot_b E_3),\ g_4,\ (g_4^1, c \cdot_c E_3)), \\
& ((g_2^1, a \cdot_a E_2 \cdot_b E_3),\ g_4,\ (g_4^1, c \cdot_c E_3)), \\
& ((g_4^1, c \cdot_c E_3),\ g_4,\ (g_4^1, c \cdot_c E_3)), \\
& (c,\ (\varepsilon^1, E_1 \cdot_a E_2 \cdot_b E_3)), \qquad (c,\ (g_4^1, c \cdot_c E_3)), \\
& (c,\ (g_1^1, a \cdot_a E_2 \cdot_b E_3)), \qquad (c,\ (g_2^1, a \cdot_a E_2 \cdot_b E_3)), \\
& (a,\ (f_3^1, a \cdot_b E_2 \cdot_b E_3)), \qquad (c,\ (f_3^2, b \cdot_b E_2 \cdot_b E_3)), \\
& \}
\end{aligned}
$$

The c-continuation tree automaton associated to $\overline{E}$ is shown in Figure 3.6

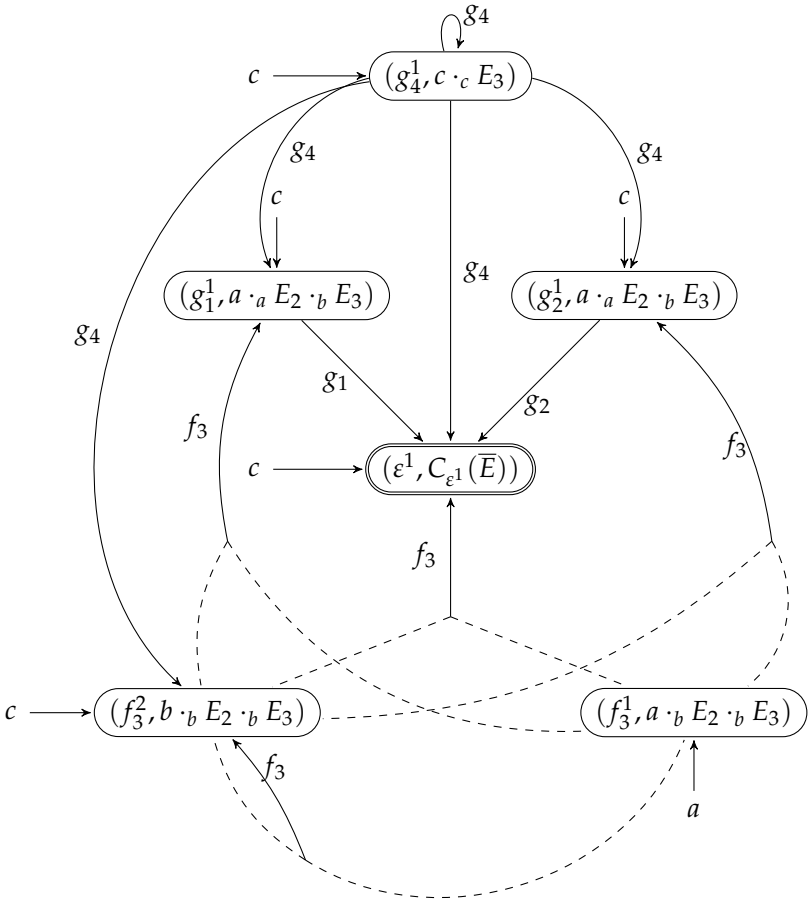


FIGURE 3.6: The C-Continuation Tree Automaton of $\overline{E}$.

3.5 Relation between the Top-Down Constructed Tree Automata

In this Section, we show the link between the constructed Top-Down tree automata shown in the previous sections.

Given $\overline{E}$, a regular tree expression over Σ and considering $\mathcal{A}_{\mathcal{C}}$ the c-continuation tree automaton of $\overline{E}$.

Quotienting $\mathcal{A}_{\mathcal{C}}$ to $\mathcal{A}_{\mathcal{E}}$ is given by *the equivalence relation* $\sim_e$ over the set of states of $\mathcal{A}_{\mathcal{C}}$ and it is defined by

$$(f_j^k, C_{f_j^k}(\overline{E})) \sim_e (g_i^p, C_{g_i^p}(\overline{E})) \Leftrightarrow h(C_{f_j^k}(\overline{E})) = h(C_{g_i^p}(\overline{E})).$$

where $(f_j^k, C_{f_j^k}(\overline{E}))$ and $(g_i^p, C_{g_i^p}(\overline{E}))$ are two states of Q of the automaton $\mathcal{A}_{\mathcal{C}}$.

This relation allows us to compute the equation tree automaton from the the c-continuation tree automaton.

Let us see an example of how to construct the equation tree automaton from the c-continuation tree automaton.

Example 3.5.1. *Let us consider the constructed position tree automaton of Example 3.1.2.*

Using the equivalence relation $\sim_e$ over the set of states of the c-continuation tree automaton.

$$h(C_{g_1^1}(\overline{E})) = h(C_{g_2^1}(\overline{E}))$$

$$h(a \cdot_a E_2 \cdot_b E_3) = h(a \cdot_a E_2 \cdot_b E_3)$$
$$\Downarrow$$
$$(g_1^1, C_{g_1^1}(\overline{E})) \sim_e (g_2^1, C_{g_2^1}(\overline{E}))$$

Thus,

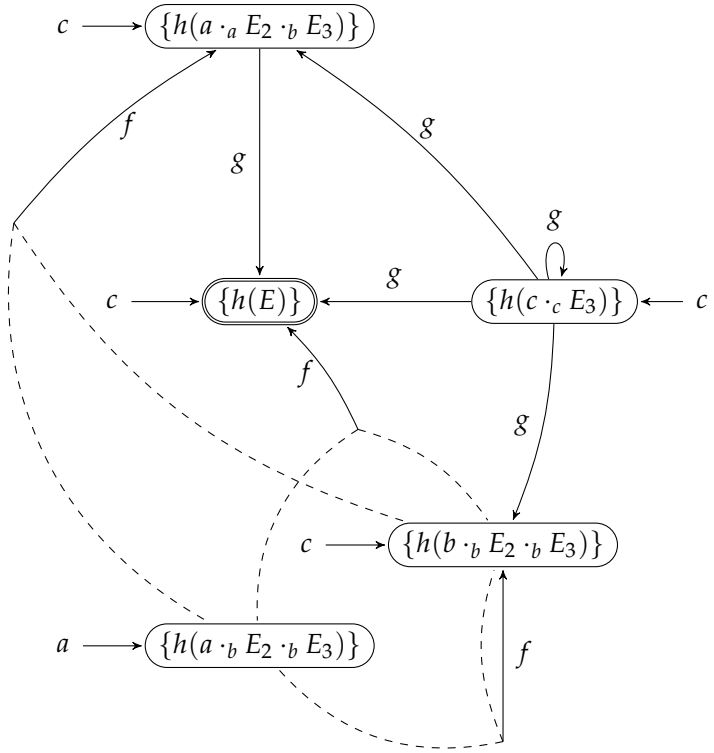


FIGURE 3.7: The Tree Automaton of $\mathcal{A}_{\mathcal{C}} / \sim_e$.

Furthermore, in order to compute the follow tree automaton from the c-continuation, we define the *similarity relation* denoted by $\equiv$ over a set of states of $\mathcal{A}_{\mathcal{C}}$ as the following:

$$(f^k, C_{f^k}(\overline{E})) \equiv (g^p, C_{g^p}(\overline{E}) \Leftrightarrow \text{Follow}(\overline{E}, f, k) = \text{Follow}(\overline{E}, g, p).$$

where $(f^k, C_{f^k}(\overline{E}))$ and $(g^p, C_{g^p}(\overline{E})$ are two states of $\mathcal{A}_{\mathcal{C}}$.

Let us see an example of a such construction.

Example 3.5.2. *Let us consider the constructed position tree automaton of Example 3.1.2.*

Using the similarity relation $\equiv$ over the set of states of the c-continuation tree automaton

$$\text{Follow}(\overline{E}, g_1, 1) = \text{Follow}(\overline{E}, g_2, 1) = \text{Follow}(\overline{E}, f_3, 2)$$
$$\Downarrow$$
$$(g_1^1, C_{g_1^1}(\overline{E})) \equiv (g_2^1, C_{g_2^1}(\overline{E})) \equiv (f_2^1, C_{g_2^1}(\overline{E}))$$

Thus,

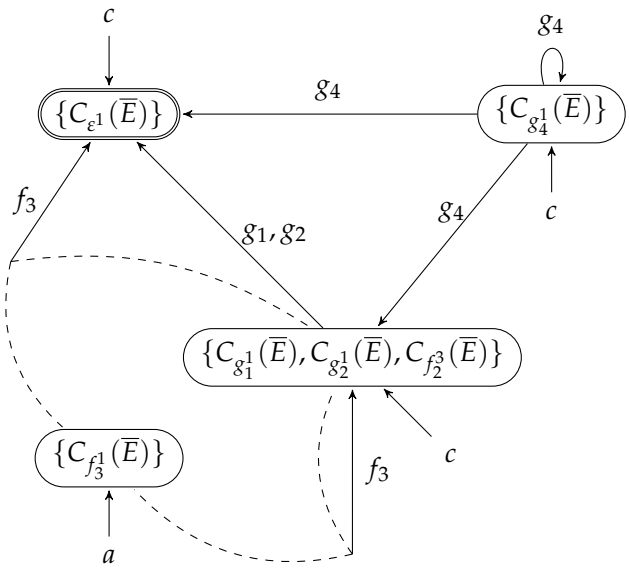


FIGURE 3.8: The Tree Automaton $\mathcal{A}_{\mathcal{C}} \ / \ \equiv$.

Lastly, the authors of [49] showed that the c-continuation and the Top-Down tree automaton are isomorphic to each other. First of all, they presented the connection between the position functions and the c-continuations. Then, they found a bijection between the set of states and a set of transitions of the both tree automata.

We can recapitulate these links in the following figure.

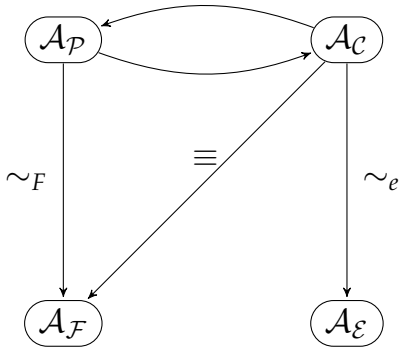


FIGURE 3.9: Relation Between Tree Automata.

3.6 Conclusion

Throughout this chapter, we have shown the transformations of a regular tree expression into tree automata, by extending the well known construction methods from words to trees in a Top-Down interpretation which recognize the same language. We have seen the construction of the position tree automata, the generalization of the follow automata, we also have presented the equation tree automata and the c-continuation tree automata.

In the following chapter, we will introduce new constructions of tree automata from a regular tree expression which are an extension of the position automata and the follow automata but upon a *Bottom-Up* interpretation.

4 The Constructions of Bottom-Up Tree Automata

We define in this chapter two new constructions of tree automata based on a Bottom-Up interpretation. Such interpretation allows the trees to start the lecture from leaves to the root.

In Section 4.1, we propose a method based on Glushkov's one using a Bottom-Up interpretation. Due to the structure of the regular expression we show in Section 4.2 how to factorize transitions of the constructed Bottom-Up tree automaton. Section 4.3 is devoted to propose a construction method of a tree automaton called *father automaton* based on the one by Ilie and Yu using a Bottom-Up approach. In the similar way as shown in Section 4.2, we show in Section 4.4 the compact version of the father automaton. Finally, in Section 4.5 we give a comparison between the Top-Down and the Bottom-Up constructions considering a family of regular tree expression.

The results of this chapter are published in [5, 6].

4.1 The Position Automaton

As we have seen in the previous chapter, Glushkov [28] showed how to construct a nondeterministic finite automaton from a given regular expression. The main idea of the construction is to define some particular sets named First (containing the symbols starting a word in the denoted languages), Follow (containing the symbols following a given symbol in the words in the denoted languages) and Last (containing the symbols ending a word in the denoted languages) that are computed with respect to the occurrences of the symbols that appear in the expression.

The Glushkov construction was extended to tree automata in a Top-Down interpretation [43, 49, 48]. The main idea of this construction is to define sets called First (containing the roots of the trees in the denoted languages), Follow (containing the symbols following a given symbol (successors) in the trees in the denoted languages) and Last (containing the leaves in the trees in the denoted languages).

Throughout this section, we suggest a construction of tree automata from a given regular tree expression based on Glushkov's one but in Bottom-Up way.

Definition 4.1.1. *The* predecessors *of a symbol f in a tree $t = f(t_1, \ldots, t_k)$ are the symbols that appear directly above it, considering that trees grow downwards.*

We denote by fathers(t, f)*, for a tree t and a symbol f the pairs*

$$\text{fathers}(t, f) = \{(g, i) \in \Sigma_l \times \mathbb{N} \mid \exists g(s_1, \ldots, s_l) \prec t, \text{root}(s_i) = f\}. \tag{4.1}$$

The Figure 4.1 shows that the fathers of the symbol f in the tree t contain the couple (g, i).

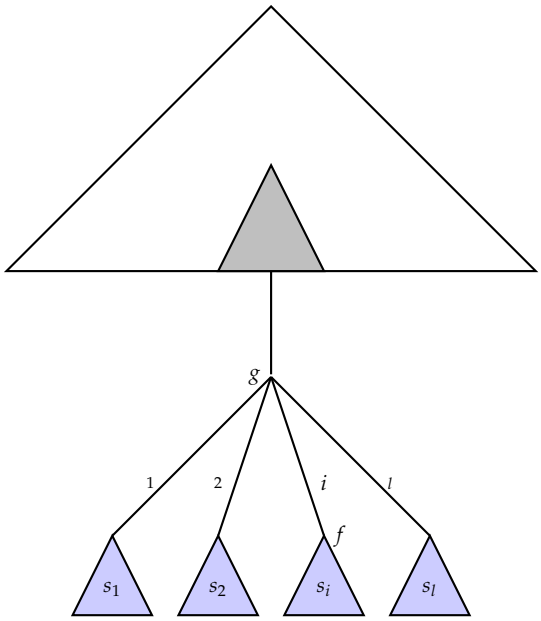


FIGURE 4.1: Illustration of fathers(t, f).

These couples link the predecessors of f and the indices of the subtrees in t whose root is f.

Let us consider a tree $t = g(t_1, \ldots, t_k)$ and a symbol f. By definition of the structure of a tree, a predecessor of f in t is a predecessor of f in a subtree t_i of t, or g if f is a root of a subtree t_i of t. Consequently:

$$\text{fathers}(t, f) = \bigcup_{i \leq n} \text{fathers}(t_i, f) \cup \{(g, i) \mid f = \text{root}(t_i)\}. \tag{4.2}$$

4.1.1 Position Functions

We define in this part the position functions that are considered in the construction of the Bottom-Up automaton. We show how to compute them and how they characterize the trees in the language denoted by a given expression.

Let $\overline{E}$ be a linear expression over a ranked alphabet Σ and f be a symbol in Σ_k.

- The set Root$(\overline{E})$, subset of Σ, contains the roots of the trees in $L(\overline{E})$, *i.e.*

$$\text{Root}(\overline{E}) = \{\text{root}(t) \mid t \in L(\overline{E})\}. \tag{4.3}$$

  Notice that this set is similar to the set First defined in the previous chapter.

- The set Fathers$(\overline{E}, f)$, subset of $\Sigma \times \mathbb{N}$, contains a couple (g, i) if there exists a tree in $L(\overline{E})$ with a node labeled by g whose i-th child is a node labeled by f:

$$\text{Fathers}(\overline{E}, f) = \bigcup_{t \in L(\overline{E})} \text{fathers}(t, f). \tag{4.4}$$

  The difference between this set and the Follow set defined in Chapter 3 resides on the sense of how one could read the next symbols of a given symbol; The successors which represent the following symbols of a given symbol (related to the set Follow) or the predecessors which represent the symbols preceding a given symbol (related to the set Fathers).

The regular tree expressions are defined inductively; Therefore, these functions can be computed in the same way over the structure of the linear expression $\overline{E}$. Let us show how to inductively compute these functions.

Lemma 4.1.1. *Let $\overline{E}$ be a linear regular expression over a ranked alphabet Σ. The set $\mathrm{Root}(\overline{E})$ is inductively computed as follows:*

$$\begin{aligned}
\mathrm{Root}(f(\overline{E}_1,...,\overline{E}_n)) &= \{f\},\\
\mathrm{Root}(\overline{E}_1 + \overline{E}_2) &= \mathrm{Root}(\overline{E}_1) \cup \mathrm{Root}(\overline{E}_2),\\
\mathrm{Root}(\overline{E}_1 \cdot_c \overline{E}_2) &= \begin{cases} \mathrm{Root}(\overline{E}_1) \setminus \{c\} \cup \mathrm{Root}(\overline{E}_2) & \text{if } c \in L(\overline{E}_1),\\ \mathrm{Root}(\overline{E}_1) & \text{otherwise,} \end{cases}\\
\mathrm{Root}(\overline{E}_1^{*_c}) &= \mathrm{Root}(\overline{E}_1) \cup \{c\},
\end{aligned}$$

where $\overline{E}_1, \ldots, \overline{E}_n$ are n regular expressions over Σ, f is a symbol in Σ_n and c is a symbol in Σ_0.

Proof. Let us consider the following cases.

1. The case when $\overline{E} = f(\overline{E}_1, \ldots, \overline{E}_n)$ is a direct consequence of Equation (2.1) and Equation (4.3).

2. Let us consider the case when $\overline{E} = \overline{E}_1 + \overline{E}_2$:

$$\begin{aligned}
\mathrm{Root}(\overline{E}_1 + \overline{E}_2) &= \{\mathrm{root}(t) \mid t \in L(\overline{E}_1 + \overline{E}_2)\}\\
&= \{\mathrm{root}(t) \mid t \in L(\overline{E}_1)\} \cup \{\mathrm{root}(t) \mid t \in L(\overline{E}_2)\}\\
&= \mathrm{Root}(\overline{E}_1) \cup \mathrm{Root}(\overline{E}_2).
\end{aligned}$$

3. Let us consider that $\overline{E} = \overline{E}_1 \cdot_c \overline{E}_2$:

$$\begin{aligned}
\mathrm{Root}(\overline{E}_1 \cdot_c \overline{E}_2) &= \{\mathrm{root}(t) \mid t \in L(\overline{E}_1) \cdot_c L(\overline{E}_2)\}\\
&= \{\mathrm{root}(t) \mid t \in t_1 \cdot_c L(\overline{E}_2),\ t_1 \in L(\overline{E}_1)\}
\end{aligned}$$

   (a) If $c \in L(\overline{E}_1)$:

$$\begin{aligned}
\mathrm{Root}(\overline{E}_1 \cdot_c \overline{E}_2) &= \{\mathrm{root}(t) \mid t \in t_1 \cdot_c L(\overline{E}_2), t_1 \in L(\overline{E}_1)\}\\
&\quad \cup \{\mathrm{root}(t) \mid t \in t_1 \cdot_c L(\overline{E}_2),\ t_1 \neq c,\ t_1 \in L(\overline{E}_1)\}\\
&= \mathrm{Root}(\overline{E}_2) \cup \mathrm{Root}(\overline{E}_1) \setminus \{c\}.
\end{aligned}$$

   (b) If $c \notin L(\overline{E}_1)$:

$$\begin{aligned}
\mathrm{Root}(\overline{E}_1 \cdot_c \overline{E}_2) &= \{\mathrm{root}(t) \mid t \in t_1 \cdot_c L(\overline{E}_2),\ t_1 \in L(\overline{E}_1)\}\\
&= \mathrm{Root}(\overline{E}_1).
\end{aligned}$$

4. Let us suppose that $\overline{E} = \overline{E}_1^{*_c}$:

$$\begin{aligned}
\mathrm{Root}(\overline{E}_1^{*_c}) =& \{\mathrm{root}(t) \mid t \in L(\overline{E}_1^{*_c})\}\\
=& \{\mathrm{root}(t) \mid t = c \in L(\overline{E}_1)\} \cup \{\mathrm{root}(t) \mid t \in t_1 \cdot_c L(\overline{E}_1^{*_c}),\ t_1 \neq c,\ t_1 \in L(\overline{E}_1)\}\\
=& \{c\} \cup \mathrm{Root}(\overline{E}_1).
\end{aligned}$$

□

Lemma 4.1.2. *Let $\overline{E}$ be a linear regular expression and f be a symbol in Σ_k. The set* $\mathrm{Fathers}(\overline{E}, f)$ *is inductively computed as follows:*

$$\begin{aligned}
\mathrm{Fathers}(g(\overline{E}_1,...,\overline{E}_n), f) &= \bigcup_{i\leq n} \mathrm{Fathers}(\overline{E}_i, f) \cup \{(g,i) \mid f \in \mathrm{Root}(\overline{E}_i)\},\\
\mathrm{Fathers}(\overline{E}_1 + \overline{E}_2, f) &= \mathrm{Fathers}(\overline{E}_1, f) \cup \mathrm{Fathers}(\overline{E}_2, f),\\
\mathrm{Fathers}(\overline{E}_1 \cdot_c \overline{E}_2, f) &= (\mathrm{Fathers}(\overline{E}_1, f) \mid f \neq c) \cup \mathrm{Fathers}(\overline{E}_2, f)\\
&\quad \cup (\mathrm{Fathers}(\overline{E}_1, c) \mid f \in \mathrm{Root}(\overline{E}_2))\\
\mathrm{Fathers}(\overline{E}_1^{*_c}, f) &= \mathrm{Fathers}(\overline{E}_1, f) \cup (\mathrm{Fathers}(\overline{E}_1, c) \mid f \in \mathrm{Root}(\overline{E}_1)),
\end{aligned}$$

where $\overline{E}_1, \ldots, \overline{E}_n$ are n regular expressions over Σ, g is a symbol in Σ_n and c is a symbol in Σ_0.

Proof. Let us consider the following cases.

1. The case when $\overline{E} = g(\overline{E}_1, \ldots, \overline{E}_n)$ is a direct consequence of Equation (4.2) and Equation (4.3).

2. Let us suppose $\overline{E} = \overline{E}_1 + \overline{E}_2$:

$$\begin{aligned}
&\mathrm{Fathers}(\overline{E}_1 + \overline{E}_2, f)\\
&= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in L(\overline{E}_1 + \overline{E}_2), \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}\\
&= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in L(\overline{E}_1), \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}\\
&\quad \cup \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in L(\overline{E}_2), \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}\\
&= \mathrm{Fathers}(\overline{E}_1, f) \cup \mathrm{Fathers}(\overline{E}_2, f).
\end{aligned}$$

3. Let us consider $\overline{E} = \overline{E}_1 \cdot_c \overline{E}_2$:

$$\begin{aligned}
&\mathrm{Fathers}(\overline{E}_1 \cdot_c \overline{E}_2, f)\\
&= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in L(\overline{E}_1) \cdot_c L(\overline{E}_2), \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}\\
&= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in t_1 \cdot_c L(\overline{E}_2), \exists g(s_1,\ldots,s_l) \prec t, t_1 \in L(\overline{E}_1),\\
&\quad \mathrm{root}(s_i) = f\}\\
&= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t_1 \in L(\overline{E}_1), c \prec t_1, \exists t_2 \in L(\overline{E}_2), g(s_1,\ldots,s_l) \prec t_2,\\
&\quad \mathrm{root}(s_i) = f\}\\
&\quad \cup \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t_1 \in L(\overline{E}_1), \exists g(s_1,\ldots,s_l) \prec t_1, s_i \neq c, \mathrm{root}(s_i) = f\}\\
&\quad \cup \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t_1 \in L(\overline{E}_1), \exists g(s_1,\ldots,s_{i-1},c,\ldots,s_l) \prec t_1, \exists s_i \in L(\overline{E}_2),\\
&\quad \mathrm{root}(s_i) = f\}\\
&= \mathrm{Fathers}(\overline{E}_2, f) \cup (\mathrm{Fathers}(\overline{E}_1, f) \mid f \neq c) \cup (\mathrm{Fathers}(\overline{E}_1, c) \mid f \in \mathrm{Root}(\overline{E}_1)).
\end{aligned}$$

4. Let us consider the case when $\overline{E} = \overline{E}_1^{*_c}$:

$$\begin{aligned}
&\mathrm{Fathers}(\overline{E}_1^{*_c}, f)\\
&= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in L(\overline{E}_1^{*_c}), \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}\\
&= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in \bigcup_{k\geq 0} L(\overline{E}_1)^{c,k}, \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}.
\end{aligned}$$

Let us set

$$S_k = \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in L(\overline{E}_1)^{c,k}, k \geq 0, \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}.$$

Let us proceed by recursion over k. **(a)** If $k = 0$ then $S_0 = \emptyset$. **(b)** If $k = 1$:

$$\begin{aligned}
&S_1\\
&\quad= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in L(\overline{E}_1)^{c,1}, \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}\\
&\quad= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in (L(\overline{E}_1) \cdot_c L(\overline{E}_1)^{c,0} \cup L(E_1)^{c,0}), \exists g(s_1,\ldots,s_l) \prec t,\\
&\qquad \mathrm{root}(s_i) = f\}\\
&\quad= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in t_1 \cdot_c \{c\} \cup \{c\}, \exists g(s_1,\ldots,s_l) \prec t, t_1 \in L(\overline{E}_1),\\
&\qquad \mathrm{root}(s_i) = f\}\\
&\quad= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in \{t_1\} \cup \{c\}, \exists g(s_1,\ldots,s_l) \prec t, t_1 \in L(\overline{E}_1),\\
&\qquad \mathrm{root}(s_i) = f\}\\
&\quad= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t \in L(\overline{E}_1), \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}\\
&\qquad \cup \{(g,i) \in \Sigma_l \times \mathbb{N} \mid \exists t = c, \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}.\\
&\quad= \mathrm{Fathers}(\overline{E}_1, f) \cup \emptyset
\end{aligned}$$

Notice that $(\mathrm{Fathers}(\overline{E}_1, c) \mid f \in \mathrm{Root}(\overline{E}_1)) \subset \mathrm{Fathers}(\overline{E}_1, c)$.

Thus,

$$S_1 = \mathrm{Fathers}(\overline{E}_1, f) \cup (\mathrm{Fathers}(\overline{E}_1, c) \mid f \in \mathrm{Root}(\overline{E}_1)).$$

(c) If $k > 1$:

$$\begin{aligned}
&S_k\\
&\quad= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid t \in (L(\overline{E}_1) \cdot_c L(\overline{E}_1)^{c,k-1} \cup L(\overline{E}_1)^{c,k-1}), \exists g(s_1,\ldots,s_l) \prec t\\
&\qquad \mathrm{root}(s_i) = f\}\\
&\quad= \{(g,i) \in \Sigma_l \times \mathbb{N} \mid t \in t_1 \cdot_c L(\overline{E}_1)^{c,k-1}, \exists g(s_1,\ldots,s_l) \prec t, t_1 \in L(\overline{E}_1),\\
&\qquad \mathrm{root}(s_i) = f\}\\
&\qquad \cup \{(g,i) \in \Sigma_l \times \mathbb{N} \mid t \in L(\overline{E}_1)^{c,k-1}, \exists g(s_1,\ldots,s_l) \prec t, \mathrm{root}(s_i) = f\}\\
&\quad= (\mathrm{Fathers}(\overline{E}_1^{c,k-1}, f) \cup (\mathrm{Fathers}(\overline{E}_1, f) \mid f \neq c) \cup (\mathrm{Fathers}(\overline{E}_1, c) \mid f \in \mathrm{Root}(\overline{E}_1)))\\
&\qquad \cup (\mathrm{Fathers}(\overline{E}_1, f) \cup (\mathrm{Fathers}(\overline{E}_1, c) \mid f \in \mathrm{Root}(\overline{E}_1)))\\
&\quad= \mathrm{Fathers}(\overline{E}_1, f) \cup (\mathrm{Fathers}(\overline{E}_1, c) \mid f \in \mathrm{Root}(\overline{E}_1)).
\end{aligned}$$

□

Example 4.1.1. *Let us consider the ranked alphabet defined by* $\Sigma_2 = \{f\}$, $\Sigma_1 = \{g\}$, *and* $\Sigma_0 = \{a,b\}$.

Let E and $\overline{E}$ be the expressions defined by

$$E = (f(a,a) + g(b))^{*_a} \cdot_b f(g(a),b), \quad \overline{E} = (f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a),b).$$

The language denoted by $\overline{E}$ is given by

$$L(\overline{E}) = \{a,\ f_1(a,a),\ f_1(f_1(a,a),a),\ f_1(a,f_1(a,a)),\ f_1(f_1(a,a),f_1(a,a)),\\ g_2(f_3(g_4(a),a))\}.$$

Hence,

$$\begin{aligned}
\mathrm{Root}(\overline{E}) &= \mathrm{Root}((f_1(a,a)+g_2(b))^{*_a} \cdot_b f_3(g_4(a),b)\\
&= \mathrm{Root}((f_1(a,a)+g_2(b))^{*_a})\\
&= \mathrm{Root}(f_1(a,a)+g_2(b)) \cup \{a\}\\
&= \mathrm{Root}(f_1(a,a)) \cup \mathrm{Root}(g_2(b)) \cup \{a\}\\
&= \{f_1, g_2, a\},
\end{aligned}$$

$$\begin{aligned}
\mathrm{Fathers}(\overline{E}, f_1) &= \mathrm{Fathers}((f_1(a,a)+g_2(b))^{*_a} \cdot_b f_3(g_4(a),b), f_1)\\
&= \mathrm{Fathers}((f_1(a,a)+g_2(b))^{*_a}, f_1) \cup \mathrm{Fathers}(f_3(g_4(a),b), f_1)\\
&= \mathrm{Fathers}(f_1(a,a)+g_2(b), f_1) \cup \mathrm{Fathers}(f_1(a,a)+g_2(b), a)\\
&\quad \cup \mathrm{Fathers}(f_3(g_4(a),b), f_1)\\
&= \mathrm{Fathers}(f_1(a,a), f_1) \cup \mathrm{Fathers}(g_2(b), f_1) \cup \mathrm{Fathers}(f_1(a,a), a)\\
&\quad \cup \mathrm{Fathers}(g_2(b), a) \cup \mathrm{Fathers}(f_3(g_4(a),b), f_1)\\
&= \{(f_1,1), (f_1,2)\},
\end{aligned}$$

$$\begin{aligned}
\text{Fathers}(\overline{E}, a) &= \text{Fathers}((f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b), a) \\
&= \text{Fathers}((f_1(a,a) + g_2(b))^{*_a}, a) \cup \text{Fathers}(f_3(g_4(a), b), a) \\
&= \text{Fathers}(f_1(a,a) + g_2(b), a) \cup \text{Fathers}(f_3(g_4(a), b), a) \\
&= \text{Fathers}(f_1(a,a), a) \cup \text{Fathers}(g_2(b), a) \cup \text{Fathers}(f_3(g_4(a), b), a) \\
&= \{(f_1, 1), (f_1, 2)\} \cup \{(g_4, 1)\} \\
&= \{(f_1, 1), (f_1, 2), (g_4, 1)\},
\end{aligned}$$

$$\begin{aligned}
\text{Fathers}(\overline{E}, g_2) &= \text{Fathers}((f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b), g_2) \\
&= \text{Fathers}((f_1(a,a) + g_2(b))^{*_a}, g_2) \cup \text{Fathers}(f_3(g_4(a), b), g_2) \\
&= \text{Fathers}(f_1(a,a) + g_2(b), g_2) \cup \text{Fathers}(f_1(a,a) + g_2(b), a) \\
&\quad \cup \text{Fathers}(f_3(g_4(a), b), g_2) \\
&= \text{Fathers}(f_1(a,a), g_2) \cup \text{Fathers}(g_2(b), g_2) \cup \text{Fathers}(f_1(a,a), a) \\
&\quad \cup \text{Fathers}(g_2(b), a) \cup \text{Fathers}(f_3(g_4(a), b), g_2) \\
&= \{(f_1, 1), (f_1, 2)\},
\end{aligned}$$

$$\begin{aligned}
\text{Fathers}(\overline{E}, b) &= \text{Fathers}((f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b), b) \\
&= \text{Fathers}(f_3(g_4(a), b), b) \\
&= \{(f_3, 2)\},
\end{aligned}$$

$$\begin{aligned}
\text{Fathers}(\overline{E}, f_3) &= \text{Fathers}((f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b), g_2) \\
&= \text{Fathers}((f_1(a,a) + g_2(b))^{*_a}, f_3) \cup \text{Fathers}(f_3(g_4(a), b), f_3) \\
&\quad \cup \text{Fathers}((f_1(a,a) + g_2(b))^{*_a}, b) \\
&= \text{Fathers}((f_1(a,a) + g_2(b))^{*_a}, b) \\
&= \text{Fathers}((f_1(a,a) + g_2(b), b) \\
&= \text{Fathers}(f_1(a,a), b) \cup \text{Fathers}(g_2(b), b) \\
&= \{(g_2, 1)\},
\end{aligned}$$

$$\begin{aligned}
\text{Fathers}(\overline{E}, g_4) &= \text{Fathers}((f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b), g_2) \\
&= \text{Fathers}((f_1(a,a) + g_2(b))^{*_a}, g_4) \cup \text{Fathers}(f_3(g_4(a), b), g_4) \\
&= \text{Fathers}(f_3(g_4(a), b), g_4) \\
&= \{(f_3, 1)\}.
\end{aligned}$$

The membership of a tree t in the language denoted by a linear expression $\overline{E}$ can be characterized by the fact that the root of each of its subtrees t' is the father of the roots of its direct subtrees, formally:

Definition 4.1.2. *Let $\overline{E}$ be a linear expression over a ranked alphabet Σ and t be a tree in T_Σ. The property $P_{\overline{E}}(t)$ is the property defined by*

$$\forall s = f(t_1, \ldots, t_n) \prec t, \forall i \leq n, (f, i) \in \text{Fathers}(\overline{E}, \text{root}(t_i)).$$

Proposition 4.1.1. *Let $\overline{E}$ be a linear expression over a ranked alphabet Σ and t be a tree in T_Σ. Then the two following conditions are equivalent:*

1. *t is in $L(\overline{E})$,*

2. $\text{root}(t)$ *is in* $\text{Root}(\overline{E})$ *and* $P_{\overline{E}}(t)$ *is satisfied.*

Proof. Let us first notice that the proposition $1 \Rightarrow 2$ is direct by definition of Root and Fathers. Let us show the second implication by induction over the structure of $\overline{E}$. Hence, let us suppose that $\text{root}(t)$ is in $\text{Root}(\overline{E})$ and $P_{\overline{E}}(t)$ is satisfied.

- Let us consider the case when $\overline{E} = g(\overline{E}_1, \ldots, \overline{E}_n)$ and let us set $t = f(t_1, \ldots, t_n)$. Since $\text{root}(t)$ is in $\text{Root}(\overline{E})$, $f = g$ from Lemma 4.1.1. From $P_{\overline{E}}(t)$, it holds that for any $i \leq n$, $(f, i) \in \text{Fathers}(\overline{E}, \text{root}(t_i))$. Since $\overline{E}$ is linear, and following Lemma 4.1.2, $\text{root}(t_i) \in \text{Root}(\overline{E}_i)$. Consequently, from the induction hypothesis, t_i is in $L(\overline{E}_i)$ for any integer $i \leq n$ and t belongs to $L(\overline{E})$.

- The case of the sum is a direct application of the induction hypothesis.

- Let us consider the case when $\overline{E} = \overline{E}_1 \cdot_c \overline{E}_2$. Let us first suppose that $\text{root}(t)$ is in $\text{Root}(\overline{E}_2)$. Then c is in $L(\overline{E}_1)$ and $P_{\overline{E}}(t)$ is equivalent to

$$\forall s = f(t_1, \ldots, t_n) \prec t, \forall i \leq n, (f, i) \in \text{Fathers}(\overline{E}_2, \text{root}(t_i)).$$

  By induction hypothesis t is in $L(\overline{E}_2)$ and therefore in $L(\overline{E})$.

  Let us suppose now that $\text{root}(t)$ is in $\text{Root}(\overline{E}_1)$. Since $\overline{E}$ is linear, let us consider the subtrees t_2 of t with only symbols of $\overline{E}_2$ and a symbol of $\overline{E}_1$ as a predecessor in t. Since $P(t)$ holds, according to induction hypothesis and Lemma 4.1.2, each of these trees belongs to $L(\overline{E}_2)$. Hence t belongs to $t_1 \cdot_c L(\overline{E}_2)$ where t_1 is equal to t where the previously defined t_2 trees are replaced by c. Once again, since $P_{\overline{E}}(t)$ holds and since $\text{root}(t)$ is in $\text{Root}(\overline{E}_1)$, t_1 belongs to $L(\overline{E}_1)$.

  In these two cases, t belongs to $L(\overline{E})$.

- Let us consider the case when $\overline{E} = \overline{E}_1^{*_c}$. Let us proceed by induction over the structure of t. If $t = c$, the proposition holds from Lemma 4.1.1 and Lemma 4.1.2. Following Lemma 4.1.2, each predecessor of a symbol f in t is a predecessor of f in $\overline{E}_1$ (case **1**) or a predecessor of c in $\overline{E}_1$ (case **2**). If all the predecessors of the symbols satisfy the case **1**, then by induction hypothesis t belongs to $L(\overline{E}_1)$ and therefore to $L(\overline{E})$. Otherwise, we can consider (similarly to the concatenation product case) the smallest subtrees t_2 of t the root of which admits a predecessor in t which is a predecessor of c in $\overline{E}_1$. By induction hypothesis, these trees belong to $L(\overline{E}_1)$. And consequently t belongs to $t' \cdots L(\overline{E}_1)$ where t' is equal to t where the subtrees t_2 have been substituted by c. Once again, by induction hypothesis, t' belongs to $L(\overline{E}_1^{*_c})$. As a direct consequence, t belongs to $L(\overline{E})$.

□

4.1.2 The Bottom-Up Position Tree Automaton

In this part, we show how to compute a Bottom-Up automaton with a linear number of states from the position functions previously defined.

Definition 4.1.3. *The* Bottom-Up position automaton $\mathcal{P}_{\overline{E}}$ *of a linear expression* $\overline{E}$ *over a ranked alphabet* Σ *is the automaton* $(\Sigma, \text{Pos}(\overline{E}), \text{Root}(\overline{E}), \delta)$ *defined by:*

$$((f_1, \ldots, f_n), g, h) \in \delta \Leftrightarrow h = g, (g, i) \in \text{Fathers}(\overline{E}, f_i), \forall i \leq n.$$

Notice that due to the linearity of $\overline{E}$, $\mathcal{P}_{\overline{E}}$ is deterministic.

Example 4.1.2. *The Bottom-Up position automaton of the expression $\overline{E}$ defined in Example 4.1.1 is defined as follows:*

The set of states is

$$\mathrm{Pos}(\overline{E}) = \{a, b, f_1, g_2, f_3, g_4\},$$

The set of final states is

$$\mathrm{Root}(\overline{E}) = \{a, f_1, g_2\},$$

The set of transition rules is

$$\begin{aligned}\delta = \{&(a,a),(b,b),((a,a),f_1,f_1),((a,f_1),f_1,f_1),((a,g_2),f_1,f_1),((f_1,a),f_1,f_1),\\ &((f_1,f_1),f_1,f_1),((f_1,g_2),f_1,f_1),((g_2,a),f_1,f_1),((g_2,f_1),f_1,f_1),\\ &((g_2,g_2),f_1,f_1),(f_3,g_2,g_2),((b,g_4),f_3,f_3),(a,g_4,g_4)\}.\end{aligned}$$

The Bottom-Up position automaton is given in Figure 4.2.

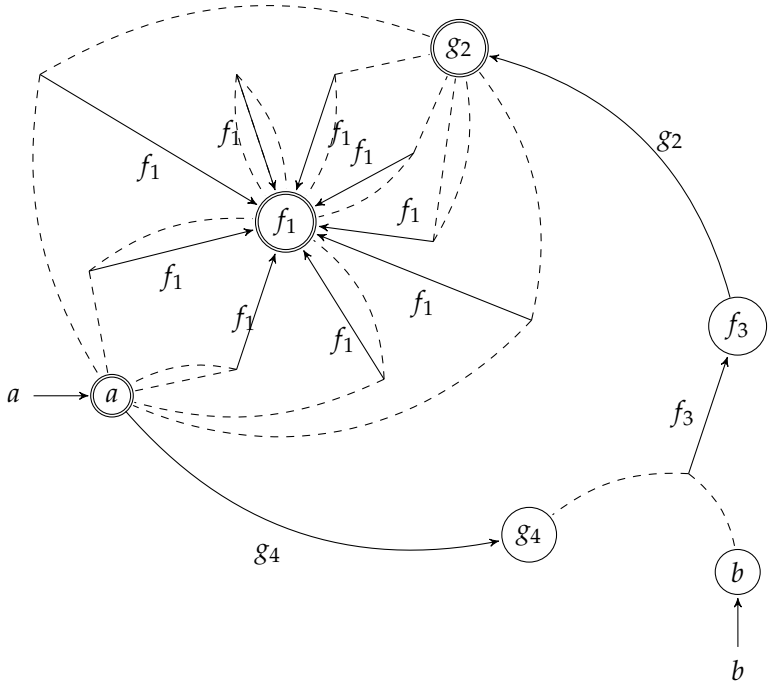


FIGURE 4.2: The Bottom-Up Position Automaton of the Expression $(f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b)$.

Let us now show that the position automaton of $\overline{E}$ recognizes $L(\overline{E})$.

Lemma 4.1.3. *Let $\mathcal{P}_{\overline{E}} = (\Sigma, Q, F, \delta)$ be the Bottom-Up position automaton of a linear regular expression $\overline{E}$ over a ranked alphabet Σ, t be a tree in T_Σ and f be a symbol in $\mathrm{Pos}(\overline{E})$. Then the two following conditions are equivalent*

1. $f \in \Delta(t)$,
2. $\mathrm{root}(t) = f \wedge P_{\overline{E}}(t)$.

Proof. Let us proceed by induction over the structure of $t = f(t_1, \ldots, t_n)$. By definition, $\Delta(t) = \delta(\Delta(t_1), \ldots, \Delta(t_n), f)$. For any state f_i in Q, it holds from the induction hypothesis that

$$f_i \in \Delta(t_i) \Leftrightarrow \mathrm{root}(t_i) = f_i \wedge P_{\overline{E}}(t_i). \qquad (*)$$

Then, suppose that **(1)** holds (*i.e.* $f \in \Delta(t)$). Equivalently, there exists by definition of $\mathcal{P}_{\overline{E}}$ and of the proposition $P_{\overline{E}}(t)$ a transition $((f_1, \ldots, f_n), f, f)$ in δ such that f_i is in $\Delta(t_i)$ for any integer $i \leq n$. Consequently, f is the root of t. Moreover, from the equivalence stated in Equation (*), $\text{root}(t_i) = f_i$ and $P_{\overline{E}}(t_i)$ holds for any integer $i \leq n$. Finally and equivalently, $P_{\overline{E}}(t)$ holds as a consequence of Equation (4.2). The reciprocal condition can be proved similarly since only equivalences are considered. □

As a direct consequence of Lemma 4.1.3 and Proposition 4.1.1,

Proposition 4.1.2. *The Bottom-Up position automaton of a linear expression $\overline{E}$ is an automaton with $|\text{Pos}(E)|$ states that recognizes $L(\overline{E})$.*

The Bottom-Up position automaton of a (not necessarily linear) expression E can be obtained by first computing the Bottom-Up position automaton of its linearized expression $\overline{E}$ and then by applying the alphabetical morphism h.

4.2 The Compressed Position Tree Automaton

In this section, we show that the structure of an expression allows us to factorize the transitions of a tree automaton by only considering the values of the Fathers function.

Indeed, let us notice that the transition structure of the Bottom-Up position automaton (for $\overline{E}$ linear) satisfies the following property: for any symbol $f \in \Sigma_k$, the set of transitions

$$\Delta_f = \delta \cap Q \times \{f\} \times \{q\}$$

has the form

$$\Delta_f = Q_1 \times Q_2 \times \ldots \times Q_k \times \{f\} \times \{q\}$$

for certain sets $Q_i \subseteq Q$ and a certain state q.

From this structure, let us show how to define a factorization method by defining a new automaton structure.

The basic idea of the factorization is to consider the cartesian product of sets. Notice that this technique has already been used to optimize the determinization of tree automata [27, 26].

Imagine that a tree automaton contains four binary transitions (q_1, q_1, f, q_3), (q_1, q_2, f, q_3), (q_2, q_1, f, q_3) and (q_2, q_2, f, q_3). These four transitions can be factorized as a *compressed transition* $(\{q_1, q_2\}, \{q_1, q_2\}, f, q_3)$ using set of states instead of sets as it is shown in Figure 4.3 and 4.4.

The compressed transitions are graphically represented as follows: each starting set is represented by dashed arcs from its elements to a second part that is classically represented as an hyper-transition.

The compressed transition $(\{q_1, q_2\}, \{q_1, q_2\}, f, q_3)$ is represented by two dashed parts (two dashed arcs for the set $\{q_1, q_2\}$ and two for $\{q_1, q_2\}$) that respectively join before representing the f-labelled part.

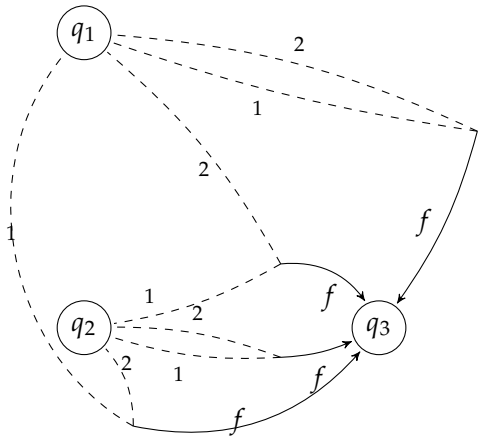


FIGURE 4.3: Before Factorization.

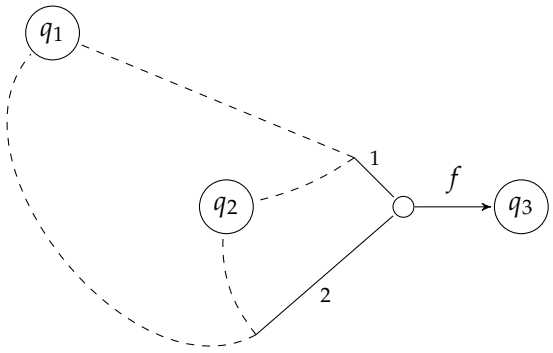


FIGURE 4.4: After Factorization.

The behavior of the original automaton can be simulated by considering the cartesian product of the origin states of the transition.

However, the notion of compression does not work for all families of automata. There is a family of automata $A_n = (\Sigma, Q, F, \Delta_f)$ where we can write Δ_f as the union of $|Q|^k$ many such sets, only by considering $Q = \{0,1\}^n$ and $(q_1, \ldots, q_k, f, q) \in \Delta_f$ if and only if

$$q_1 \oplus q_2 \oplus q_3 \ldots \oplus q_k \oplus q = 0$$

where q_i is a n-bit vector and $\oplus$ is the bit-wise exclusive or.

We first show how to encode such a notion of compressed automaton and how it can be used in order to solve the membership test.

Definition 4.2.1. *A* compressed tree automaton *over a ranked alphabet Σ is a 4-tuple (Σ, Q, F, δ) where Q is a set* of states, *$F \subset Q$ is the set of* final states, *$\delta \subset (2^Q)^n \times \Sigma_n \times Q$ is the set of* compressed transitions *that can be seen as a function from $(2^Q)^k \times \Sigma_k$ to 2^Q defined by*

$$(Q_1, \ldots, Q_k, f, q) \in \delta \Leftrightarrow q \in \delta(Q_1, \ldots, Q_k, f)$$

Example 4.2.1. *Let us consider the compressed automaton $A = (\Sigma, Q, F, \delta)$ shown in Figure 4.5. Its transitions are*

$$\delta = \{(\{1,2,5\}, \{3,4\}, f, 1), (\{2,3,5\}, \{4,6\}, f, 2), \\ (\{1,2\}, \{3\}, f, 5), (\{6\}, g, 4), (\{6\}, g, 5), (a, 6), (a, 4), (b, 3)\}.$$

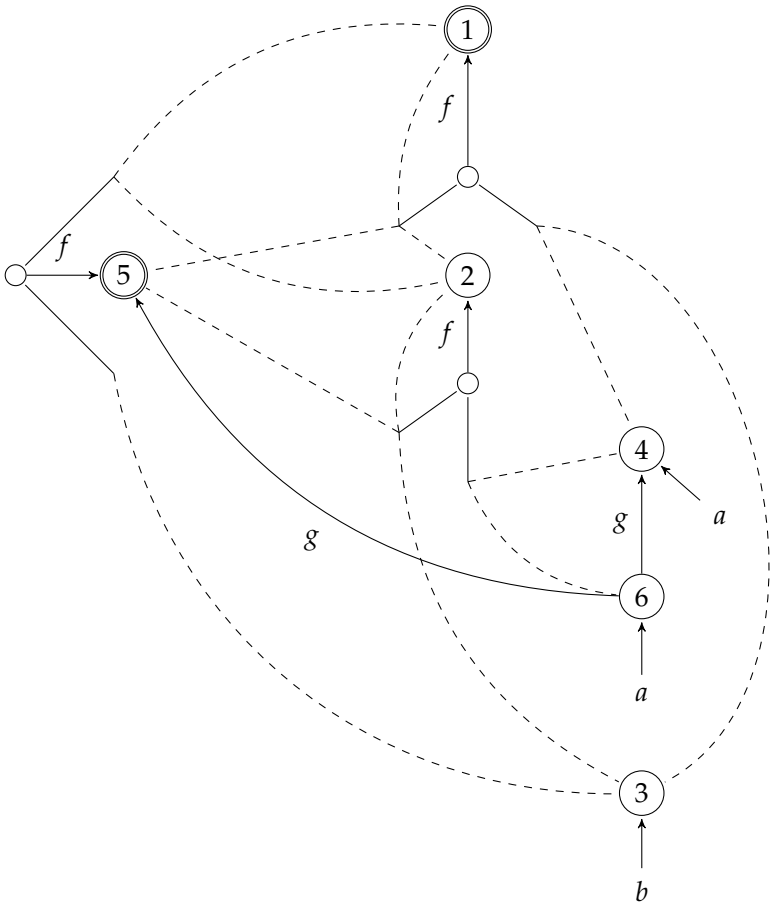


FIGURE 4.5: The Compressed Automaton A.

The transition function δ can be restricted to a function from $Q^n \times \Sigma_n$ to 2^Q (*e.g.* in order to simulate the behavior of an uncompressed automaton) by considering for a tuple $(q_1, \ldots, q_k)$ of states and a symbol f in Σ_k all the "active" transitions $(Q_1, \ldots, Q_k, f, q)$, that are the transitions where q_i is in Q_i for $i \leq k$. More formally, for any k states $(q_1, \ldots, q_k)$ in Q^k, for any symbol f in Σ_k,

$$\delta(q_1, \ldots, q_k, f) = \bigcup_{\substack{(Q_1,\ldots,Q_k,f,q)\in\delta,\\ \forall i\leq k, q_i\in Q_i}} \{q\}. \tag{4.5}$$

The transition set δ can be extended to a function Δ from T_Σ to 2^Q by inductively considering, for a tree $f(t_1, \ldots, t_k)$ the "active" transitions $(Q_1, \ldots, Q_k, f, q)$ once a subtree is read, that is when $\Delta(q_i)$ and Q_i admits a common state for $i \leq k$. More formally, for any tree $t = f(t_1, \ldots, t_k)$ in T_Σ,

$$\Delta(t) = \bigcup_{\substack{(Q_1,\ldots,Q_k,f,q)\in\delta,\\ \forall i\leq k, \Delta(t_i)\cap Q_i\neq\emptyset}} \{q\}.$$

As a direct consequence of the two previous equations,

$$\Delta(f(t_1, \ldots, t_n)) = \bigcup_{(q_1,\ldots,q_n)\in\Delta(t_1)\times\cdots\times\Delta(t_n)} \delta(q_1, \ldots, q_n, f). \tag{4.6}$$

The *language recognized by* a compressed automaton $A = (\Sigma, Q, F, \delta)$ is the subset $L(A)$ of T_Σ defined by

$$L(A) = \{t \in T_\Sigma \mid \Delta(t) \cap F \neq \emptyset\}.$$

Example 4.2.2. *Let us consider the automaton of Figure 4.5 and let us show that the tree* $t = f(f(b,a), g(a))$ *belongs to* $L(A)$.

In order to do so, let us compute $\Delta(t')$ *for each subtree* t' *of* t. *First, by definition,*

$$\Delta(a) = \{4,6\}, \qquad \Delta(b) = \{3\}.$$

Since the only transition in δ *labeled by* f *containing* 3 *in its first origin set and* 4 *or* 6 *in its second is the transition* $(\{2,3,5\}, \{4,6\}, f, 2)$,

$$\Delta(f(b,a)) = \{2\}.$$

Since the two transitions labeled by g *are* $(\{6\}, g, 4)$ *and* $(\{6\}, g, 5)$,

$$\Delta(g(a)) = \{4,5\}.$$

Finally, there are two transitions labeled by f *containing* 2 *in their first origin and* 4 *or* 5 *in its second:* $(\{2,3,5\}, \{4,6\}, f, 2)$ *and* $(\{1,2,5\}, \{3,4\}, f, 1)$. *Therefore*

$$\Delta(f(f(b,a), g(a))) = \{1,2\}.$$

Finally, since 1 *is a final state,* $t \in L(A)$.

Let ϕ be an alphabetical morphism between two alphabets Σ and Σ'. The *image by* ϕ of a compressed automaton $A = (\Sigma, Q, F, \delta)$ is the compressed automaton $\phi(A) = (\Sigma', Q, F, \delta')$ where

$$\delta' = \{(Q_1, \ldots, Q_n, \phi(f), q) \mid (Q_1, \ldots, Q_n, f, q) \in \delta\}.$$

By a trivial induction over the structure of the trees, it can be shown that

$$L(\phi(A)) = \phi(L(A)). \tag{4.7}$$

Due to their inductive structure, regular expressions are naturally factorizing the structure of transitions of a position automaton.

Let us now define the compressed position automaton of an expression.

Definition 4.2.2. *The* compressed Bottom-Up position automaton $\mathcal{C}(\overline{E})$ *of a linear expression* $\overline{E}$ *is the automaton* $(\Sigma, Q, \mathrm{Root}(\overline{E}), \delta)$ *defined by*

$$\delta = \{(Q_1, \ldots, Q_k, f, \{f\}) \mid Q_i = \{g \mid (f,i) \in \mathrm{Fathers}(\overline{E}, g)\}\}.$$

Example 4.2.3. *Let us consider the expression* $\overline{E}$ *defined in Example 4.1.1. The compressed automaton of* $\overline{E}$ *is represented at Figure 4.6. Its transitions are*

$$\begin{aligned}\delta = {} & (a, \{a\}),\ (b, \{b\}),\ (\{a, f_1, g_2\},\ \{a, f_1, g_2\}, f_1, \{f_1\}),\\ & (\{g_4\}, \{b\}, f_3, \{f_3\}),\ (\{f_3\}, g_2, \{g_2\}),\ (\{a\}, g_4, \{g_4\}).\end{aligned}$$

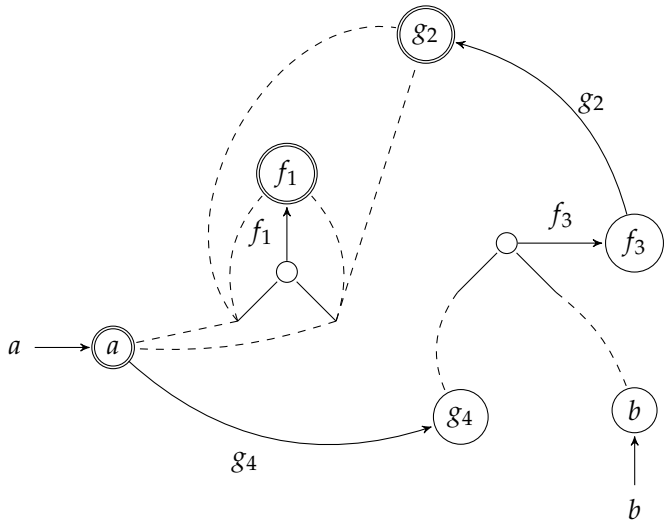


FIGURE 4.6: The Compressed Automaton of the Expression $(f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a),b)$.

As a direct consequence of Definition 4.2.2 and of Equation (4.5),

Lemma 4.2.1. *Let $\overline{E}$ be a linear expression over a ranked alphabet Σ. Let $\mathcal{C}(\overline{E}) = (\Sigma, Q, F, \delta)$. Then, for any n states $(q_1, \ldots, q_n)$ in Q^n, for any symbol f in Σ_k,*

$$\delta(q_1, \ldots, q_n, f) = \{f\} \Leftrightarrow \forall i \leq n, (f,i) \in \mathrm{Fathers}(\overline{E}, q_i).$$

Consequently, considering Definition 4.1.3, Lemma 4.2.1 and Equation (4.6),

Proposition 4.2.1. *Let $\overline{E}$ be a linear expression over a ranked alphabet Σ. Let $\mathcal{P}_{\overline{E}} = (\_, \_, \_, \delta)$ and $\mathcal{C}(\overline{E}) = (\_, \_, \_, \delta')$. For any tree t in T_Σ,*

$$\Delta(t) = \Delta'(t).$$

Since the Bottom-Up position automaton of a linear expression $\overline{E}$ and its compressed version have the same states and the same final states,

Corollary 4.2.1. *The position automaton of an expression and its compact version recognize the same language.*

The compressed Bottom-Up position automaton of a (not necessarily linear) expression E can be obtained by first computing the compressed Bottom-Up position automaton of its linearized expression $\overline{E}$ and then by applying the alphabetical morphism h. Therefore, considering Equation (4.7),

Proposition 4.2.2. *The compressed Bottom-Up position automaton of a regular expression E is an automaton with $|\mathrm{Pos}(E)|$ states and $|(\mathrm{Pos}(E)|$ transitions that recognizes $L(E)$.*

4.3 The Father Automaton

In this section, we define the *father automaton* associated with an expression which is an extension of the classical follow (word) automaton [35] that has already been extended in the case of Top-Down tree automata [49].

We embed the computation of the function Root in the function Fathers by adding a unary symbol \$ which is not in Σ located at the top of the syntactic tree of an expression.

Indeed,

$$f \in \text{Root}(E) \Leftrightarrow (\$, 1) \in \text{Fathers}(\$(E), f). \tag{4.8}$$

Equivalently,

Lemma 4.3.1. *Let f be a state of the position automaton of a linear expression $\overline{E}$. The two following conditions are equivalent:*

1. *f is a final state,*
2. *$(\$, 1)$ is in* $\text{Fathers}(\$(\overline{E}), f)$.

With this notation, the father automaton is defined as follows.

Definition 4.3.1. *The* father automaton *of a linear expression $\overline{E}$ over an alphabet Σ is the automaton $\mathcal{F}_{\overline{E}} = (\Sigma, Q, F, \delta)$ defined by*

$$Q = \{\text{Fathers}(\$(\overline{E}), f) \mid f \in \Sigma\}, \quad F = \{q \in Q \mid (\$, 1) \in q\},$$
$$((\text{Fathers}(\$(\overline{E}), f_1), \ldots, \text{Fathers}(\$(\overline{E}), f_n)), g, \text{Fathers}(\$(\overline{E}), h)) \in \delta$$
$$\Leftrightarrow h = g, \forall i \leq n, (g, i) \in \text{Fathers}(\overline{E}, f_i).$$

Notice that due to the linearity of $\overline{E}$, $\mathcal{F}_{\overline{E}}$ is deterministic.

In the word case, it has been shown that the follow automaton is a quotient of the position automaton [35].

Let us proceed in the same way in order to show that the father automaton of an expression E recognizes $L(E)$. Consequently, let us first recall how to extend the notion of congruence for tree automata, considering the notion of forward bisimulations [33].

Let $A = (\Sigma, Q, F, \delta)$ be a deterministic tree automaton and $\sim$ be an equivalence relation over Q included in $(F \times F) \cup (Q \setminus F) \times (Q \setminus F)$. Given an equivalence relation $\sim$ over a set S, we denote by $S_\sim$ the set of equivalence classes of S and by $[p]_\sim$ (or $[p]$ when there is no ambiguity) the equivalence class of an element p in P.

The relation $\sim$ is a *forward bisimulation* [33] for δ if and only if for any two equivalent states p and p' in Q, it holds : for any symbol f in Σ_m, for any integer $n \leq m$, for any $m-1$ states $q_1, \ldots, q_{n-1}, q_{n+1}, \ldots, q_m$ in Q,

$$\delta((q_1, \ldots, q_{n-1}, p, q_{n+1}, \ldots, q_m), f) \sim \delta((q_1, \ldots, q_{n-1}, p', q_{n+1}, \ldots, q_m), f).$$

According to [1], two congruent states are said to be *interchangeable*.

The *quotient automaton* of A w.r.t. $\sim$ is the automaton $A_\sim = (\Sigma, Q_\sim, F_\sim, \delta_\sim)$ with

$$\delta_\sim(([q_1], \ldots, [q_m]), f) = \{[q] \mid q \in \delta((q_1, \ldots, q_m), f)\}. \tag{4.9}$$

Proposition 4.3.1 ([33])**.** *For any tree automaton A, for any forward bisimulation $\sim$, $A_\sim$ recognizes $L(A)$.*

Let us now show how to obtain the father automaton by quotienting the position automaton w.r.t. the following congruence.

Definition 4.3.2. *The* father congruence *associated with a linear expression $\overline{E}$ over an alphabet Σ is the congruence $\sim$ defined for any symbols p,p' in Σ by*

$$p \sim p' \Leftrightarrow \text{Fathers}(\$(E), p) = \text{Fathers}(\$(E), p').$$

Proposition 4.3.2. *The father congruence of a linear expression $\overline{E}$ is a forward bisimulation for the transition function of the position automaton of $\overline{E}$.*

Proof. First, let us notice that following Lemma 4.3.1, two equivalent states have the same finality. Moreover, two states are equivalent if and only if they admit the same fathers. Consequently, from the construction of the position automaton (Definition 4.1.3), for any two equivalent states p and $p' \in Q$, it holds: for any symbol f in Σ_m, for any integer $n \leq m$, for any $m-1$ states $q_1, \ldots, q_{n-1}, q_{n+1}, \ldots, q_m$ in Q,

$$\delta((q_1, \ldots, q_{n-1}, p, q_{n+1}, \ldots, q_m), f) = \delta((q_1, \ldots, q_{n-1}, p', q_{n+1}, \ldots, q_m), f).$$

Since $\sim$ is reflexive, it is a forward bisimulation. □

Proposition 4.3.3. *The father automaton associated with a linear expression $\overline{E}$ is isomorphic to the quotient of the position automaton of $\overline{E}$ w.r.t. the father congruence.*

Proof. Let us set

$$\mathcal{P}_{\overline{E}} = (\_, \_, \_, \delta), \qquad \mathcal{P}_{\overline{E}_\sim} = (\_, \_, \_, \delta_\sim) \qquad \mathcal{F}_{\overline{E}} = (\_, \_, \_, \delta').$$

Let us consider the functions ϕ and ϕ' defined as follows

$$\phi([f]) = \text{Fathers}(\$(\overline{E}), f), \qquad \phi'(\text{Fathers}(\$(\overline{E}), f)) = [f].$$

Notice that since $f \sim f' \Leftrightarrow \text{Fathers}(\$(\overline{E}), f) = \text{Fathers}(\$(\overline{E}), f')$, the functions are both well-defined. Moreover, they are trivially the inverse of each other. Furthermore, since

$$f \sim f' \Rightarrow ((\$, 1) \in \text{Fathers}(\$(\overline{E}), f) \Leftrightarrow (\$, 1) \in \text{Fathers}(\$(\overline{E}), f')),$$

ϕ preserves the finality of the states. Finally,

$$\begin{aligned}
&(([f_1], \ldots, [f_n]), g, [g]) \in \delta_\sim \\
&\overset{\text{(Equation (4.9))}}{\Leftrightarrow} ((f_1, \ldots, f_n), g, g) \in \delta \\
&\overset{\text{(Definition 4.1.3)}}{\Leftrightarrow} \forall i \leq n, (g, i) \in \text{Fathers}(\overline{E}, f_i) \\
&\overset{\text{(Definition 4.3.1)}}{\Leftrightarrow} ((\text{Fathers}(\$(\overline{E}), f_1), \ldots, \text{Fathers}(\$(\overline{E}), f_n)), g, \text{Fathers}(\$(\overline{E}), g)) \in \delta'.
\end{aligned}$$

Hence ϕ and ϕ' are two inverse automata morphisms between $\mathcal{P}_{\overline{E}_\sim}$ and $\mathcal{F}_{\overline{E}}$. □

As a direct consequence of Proposition 4.3.1, Proposition 4.3.2 and Proposition 4.3.3,

Corollary 4.3.1. *The father automaton associated with a linear expression $\overline{E}$ recognizes $L(\overline{E})$.*

Applying the delinearization morphism h from $\mathcal{F}_{\overline{E}}$ produces the father automaton of any expression E. Finally, from Equation (2.13),

Corollary 4.3.2. *The father automaton associated with an expression E recognizes $L(E)$.*

Example 4.3.1. *The father automaton $(\text{Pos}(\overline{E}), \text{Pos}(\overline{E})_\sim, \text{Root}(\overline{E})_\sim, \delta)$ of the expression $\overline{E}$ defined in Example 4.1.1 is obtained by merging the states f_1 and g_2 of $\mathcal{P}_E$,* i.e.*:*

$$\begin{gathered}
\text{Pos}(E) = \{[a], [b], \{f_1, g_2\}, [f_3], [g_4]\}, \quad \text{Root}(\overline{E}) = \{[a], [f_1]\}, \\
\delta = \{(a, [a]), (b, [b]), (([a], [a]), f_1, [f_1]), (([a], [f_1]), f_1, [f_1]), (([f_1], a), f_1, [f_1]), \\
(([f_1], [f_1]), f_1, [f_1]), ([f_3], g_2, [g_2]), (([b], [g_4]), f_3, [f_3]), ([a], g_4, [g_4])\}.
\end{gathered}$$

The Bottom-Up father automaton is given in Figure 4.7.

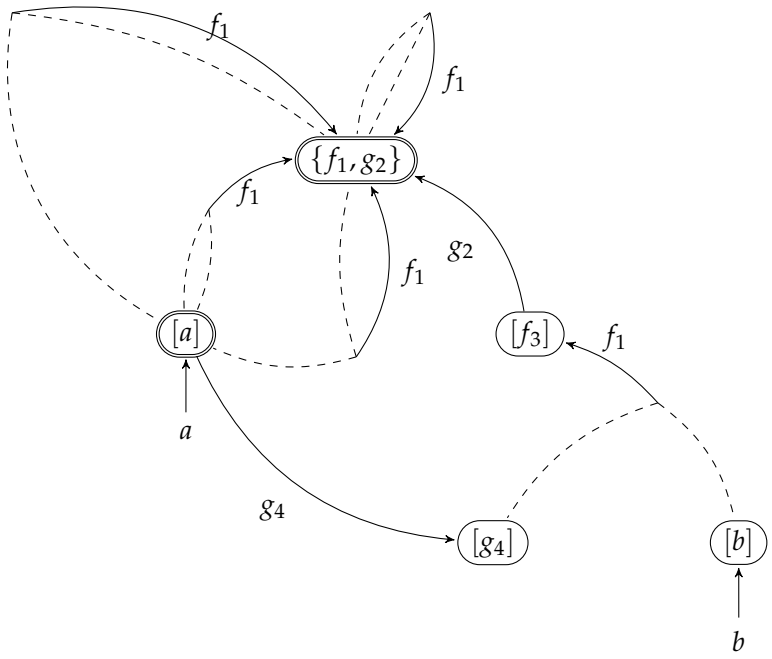


FIGURE 4.7: The Bottom-Up Father Automaton of the Expression $(f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b)$.

4.4 The Compressed Father Tree Automaton

We show in this section that the father automaton can be compressed too in the same way as used with the position automaton.

Definition 4.4.1. *The* compressed father automaton $\mathcal{CF}(\overline{E})$ *of a linear expression* $\overline{E}$ *is the automaton* $(\Sigma, \mathrm{Pos}(\overline{E}), \mathrm{Root}(\overline{E}), \delta)$ *defined by*

$$\delta = \{(Q_1, \ldots, Q_k, f, \{\mathrm{Fathers}(\$(\overline{E}), f)\}) \mid Q_i = \{\mathrm{Fathers}(\$(\overline{E}), g) \mid (f, i) \in \mathrm{Fathers}(\$(\overline{E}), g)\}\}.$$

In order to show that $\mathcal{CF}(\overline{E})$ recognizes $L(\overline{E})$, we can apply the same method as for the father automaton. First, due to Equation 4.5, the definition of a forward bisimulation for A is exactly the same (Equation (4.3)).

The *quotient* of a compressed automaton $A = (\Sigma, Q, F, \delta)$ w.r.t. a forward bisimulation $\sim$ is the compressed automaton $A_\sim = (\Sigma, Q_\sim, F_\sim, \delta')$ where

$$\delta'((Q_1, \ldots, Q_m), f) = \{\phi(q) \mid q \in \delta((q_1, \ldots, q_m), f) \wedge \forall i \leq m, [q_i] \in Q_i\}.$$

Similarly to Lemma 4.2.1 and Proposition 4.2.1, it can be shown that

Proposition 4.4.1. *The compressed father automaton is a quotient of the compressed position automaton w.r.t. the father congruence.*

As a direct corollary,

Corollary 4.4.1. *The compressed father automaton and the father automaton of a linear expression* $\overline{E}$ *recognize* $L(\overline{E})$.

Applying the delinearization morphism h from $\mathcal{CF}_{\overline{E}}$ produces the compressed father automaton of any expression E. Finally, according to Equation (4.7),

Proposition 4.4.2. *The compressed father automaton and the father automaton of an expression E recognize $L(E)$.*

Example 4.4.1. *Let us consider the expression $\overline{E}$ defined in Example 4.1.1. The compressed father automaton of $\overline{E}$ is represented by Figure 4.8.*

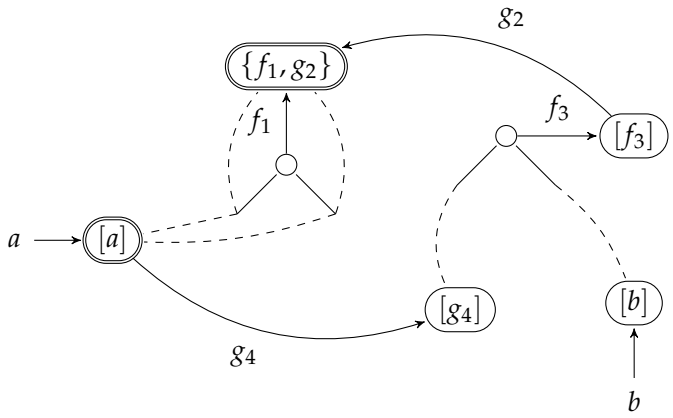


FIGURE 4.8: The Compressed Father Automaton of the Expression $(f_1(a,a) + g_2(b))^{*_a} \cdot_b f_3(g_4(a), b)$.

4.5 Comparison with the Top-Down Automata

In this section, we exhibit a parameterized family of expressions in order to compare the size of the Top-Down and Bottom-Up position automata.

Let us now consider the expressions

$$E_n = (\sum_1^n g(a))^{*_a} \cdot_a (\sum_1^n f(a,a)), \qquad \overline{E_n} = (\sum_{i=1}^n g_i(a))^{*_a} \cdot_a (\sum_{i=1}^n f_{n+i}(a,a)).$$

It can be shown that

$$\begin{aligned}
\mathrm{Root}(\overline{E_n}) &= \{g_1, \ldots, g_n, f_{n+1}, \ldots, f_{2n}\},\\
\mathrm{Fathers}(\overline{E_n}, f_) &= \mathrm{Fathers}(\overline{E_n}, g_)\\
&= \{(g_1,1), \ldots, (g_n,1)\},\\
\mathrm{Fathers}(\overline{E_n}, a) &= \{(f_{n+1},1), (f_{n+1},2), \ldots, (f_{2n},1), (f_{2n},2)\},\\
\mathrm{Follow}(\overline{E_n}, g_, 1) &= \{g_1, \ldots, g_n, f_{n+1}, \ldots, f_{2n}\},\\
\mathrm{Follow}(\overline{E_n}, f_, 1) &= \mathrm{Follow}(\overline{E_n}, f_, 2)\\
&= \{a\}.
\end{aligned}$$

Therefore,

Automata	# states	# transitions	Figure
Top-Down Position Tree Automaton	$3n+1$	$2n^2+4n$	4.9
Bottom-Up Position Tree Automaton	$2n+1$	$2n^2+n+1$	4.10
Top-Down Follow Tree Automaton	2	3	4.12
Bottom-Up Father Tree Automaton	2	3	4.12
Compressed Bottom-Up Position Tree Automaton	$2n+1$	$2n+1$	4.11
Compressed Bottom-Up Father Tree Automaton	2	3	4.12

TABLE 4.1: Comparison between Top-Down and Bottom-Up Tree Automata.

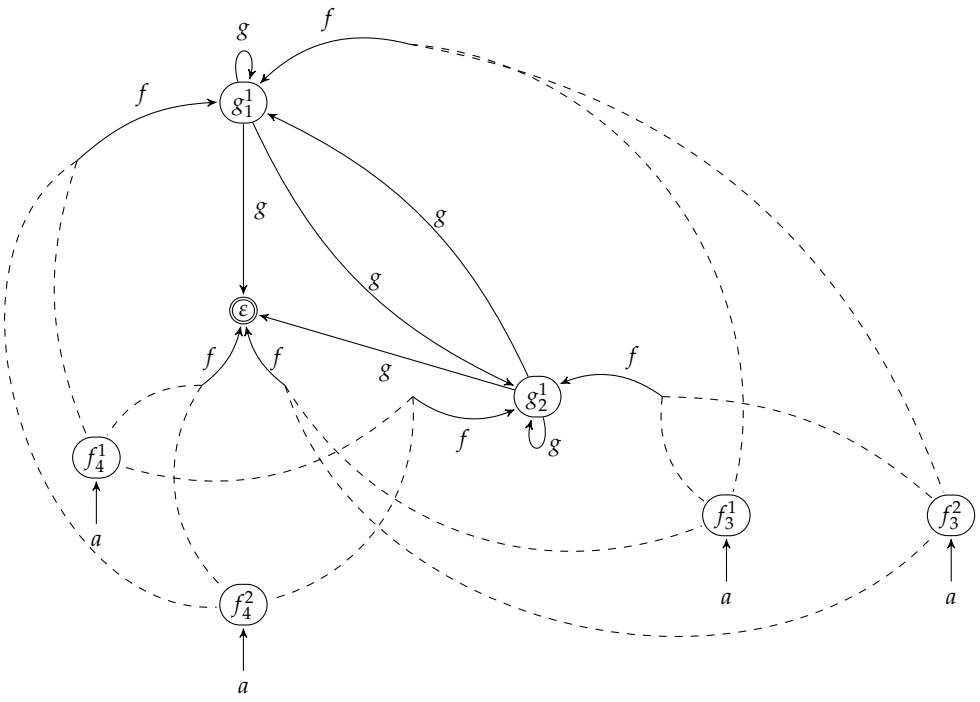


FIGURE 4.9: The Top-Down Position Automaton of E_2.

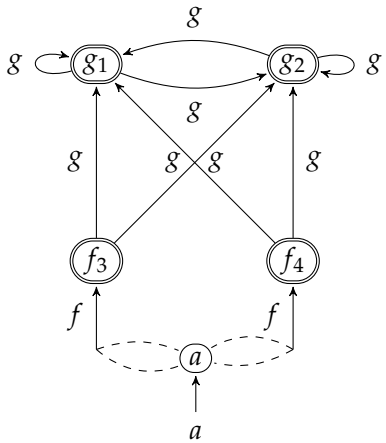


FIGURE 4.10: The Bottom Up Position Automaton of E_2.

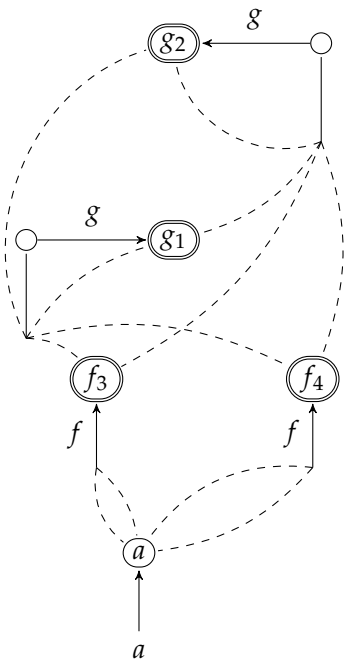


FIGURE 4.11: The Compressed Bottom Up Position Automaton of E_2.

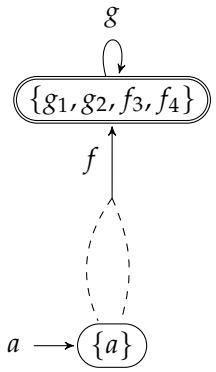


FIGURE 4.12: The Father Automaton of E_2.

This family shows that the Bottom-Up position automaton can be smaller than the Top-Down position automaton, and that the father automaton can be smaller than the compressed version of the Bottom-Up position automaton.

However, to confirm it in the general case we have to carry out a thorough study. For example, we have to find a generator of regular tree expressions in order to make a comparative study on average.

4.6 Conclusion

In this chapter, we have shown how to compute the Bottom-Up position automaton and the father automaton associated with a regular expression respectively in Section 4.1 and Section 4.3. These constructions are relatively similar to the classical ones defined over a word expression [28, 35]. Moreover, we have defined a factorization method by considering the "cartesian product" to compress the transitions of the constructed Bottom-Up automata. Furthermore, we showed that the father automaton is the quotient of the Bottom-Up position automaton and this relation still valid in their compact versions.

Finally, a comparative study between Top-Down position automaton, Bottom-Up position automaton, father automaton and their compact versions has been established by considering a family of regular tree expressions.

In Chapter 5, we extend Brzozowski derivatives [13] and Antimirov partial derivatives [2] in a Bottom-Up way (in a different way from [42]), using the Bottom-Up quotient defined in [49].

5 Bottom-Up Derivatives

In this chapter, we extend the notion of word derivatives due to Brzozowski to tree derivatives using already known inductive formulae of quotients defined in [19]. In Section 5.1, we define some restrictions of the well known Boolean operations. Section 5.2 is dedicated to introduce *the extended tree expressions* after adding some restrictions to the classical tree expressions. The construction of the derivative tree automaton from a given extended tree expression is given in Section 5.3. The results of this chapter are published in [4].

5.1 Boolean Operations

In this section, we will consider particular restrictions of Boolean operations such as union, intersection, complement, *etc.*, based on the combination of homogeneous languages with the same ε-indices.

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

$$\neg L = \{t \in T_{\Sigma_k} \mid t \notin L, \mathrm{Ind}_\varepsilon(t) = \mathrm{Ind}_\varepsilon(L)\}. \tag{5.1}$$

The computation of the Bottom-Up quotient of a complemented language is given as follows.

Proposition 5.1.1. *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_Σ 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 (2.7), we get the following result.

Corollary 5.1.1. *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_Σ*

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

5.2 Extended Tree Expressions

In this section, we integrate the homogeneous Boolean operations to tree expressions in order to obtain *the extended homogeneous tree expressions* and then we show how to compute them.

Imagine that we want to determine whether ε_j belongs to a given language or not: for instance, how can we decide whether ε_j belongs to $\neg\emptyset$ or not? The complementation function needs to know the ε-index set of the language it complements. As a solution, we can index all the occurrences of the empty set by the set of ε-indices.

Definition 5.2.1. *An* extended homogeneous tree expression *(tree expression for short) E over Σ is inductively defined by*

$$\begin{array}{lll} E = f(E_1, \ldots, E_n), & E = \varepsilon_j, & E = \emptyset_{\mathcal{I}}, \\ E = \mathrm{op}(E_1, \ldots, E_n), & E = E' \circ (E_1, \ldots, E_n), & E = E_1^{\circledast}, \\ E = E_1 \cdot_a E_2, & E = E_1^{*_a}, & \end{array} \tag{5.2}$$

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 a n-ary Boolean operator and a is a symbol in Σ_0. We denote the expression $\emptyset_\emptyset$ by $\emptyset$.*

The ε-indices set $\mathrm{Ind}_\varepsilon(E)$ of a tree expression E, that we use to distinguish tree expressions, is inductively defined, following Equation (5.2), 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; The tree expressions defined above does not guarantee that the language denoted by the tree expression is homogeneous. That is why we define the notion of *valid tree expression*, that rejects non-homogeneous tree expressions. For example, the expression $E_1 \cdot_a E_2$ is valid if and only if the expressions E_1 and E_2 are valid and E_2 is 0-homogeneous tree expression. Moreover, the expression $E_1^{\circledast}$ is valid if and only if E_1 is valid and it is 1-homogeneous tree expression (*i.e.* $|ind_\varepsilon(E_1)| = 1$) and so on. More formally:

Definition 5.2.2. *A tree expression E is* valid *if it satisfies the predicate $V(E)$ inductively defined, following Equation (5.2), by*

$$\begin{aligned} V(\varepsilon_j) = V(\emptyset_{\mathcal{I}}) &= \mathrm{True}, \\ V(f(E_1, \ldots, E_n)) &= \Big(\bigwedge_{1 \le k \le n} V(E_k)\Big) \wedge \Big(\bigwedge_{1 \le k < k' \le n} (\mathrm{Ind}_\varepsilon(E_k) \cap \mathrm{Ind}_\varepsilon(E_{k'}) = \emptyset)\Big), \\ V(\mathrm{op}(E_1, \ldots, E_n)) &= \Big(\bigwedge_{1 \le k \le n} V(E_k)\Big) \wedge \Big(\bigwedge_{1 \le k < n} (\mathrm{Ind}_\varepsilon(E_k) = \mathrm{Ind}_\varepsilon(E_{k+1}))\Big), \end{aligned}$$

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

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 a n-ary Boolean operator, op$'$ is a n-ary Boolean operation over homogeneous languages with $\mathcal{I}$ as ε-index set (*e.g.* *Equation* (5.1)) 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.

The difference between the derivation that we will present and the one introduced by Kuske and Meinecke [42] resides on the direction of this operation: instead of deriving from root to leaves we delete symbols from leaves to root and an internal node is omitted if and only if their sons are deleted.

Definition 5.2.3. *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 5.2.4. *Let α be a symbol in Σ_n and F be a valid tree expression over Σ containing $\{1, \ldots, n\}$ as ε-indices. The* derivative *of F w.r.t. to α is the expression inductively defined by*

$$\begin{aligned}
d_\alpha(\emptyset_{\mathcal{I}}) &= \emptyset_{\{1\} \cup \{i+1 | i > n, i \in \mathcal{I}\}}, \\
d_\alpha(\varepsilon_1) &= \emptyset_{\{1\}}, \\
d_\alpha(\alpha(\varepsilon_1, \ldots, \varepsilon_n)) &= \varepsilon_1, \\
d_\alpha(f(E_1, \ldots, E_m)) &= \sum_{1 \leq j \leq n} f(\underline{E_1}, \ldots, \underline{E_{j-1}}, d_\alpha(t_j), \underline{E_{j+1}}, \ldots, \underline{E_m}), \\
&\quad + \varepsilon_1 \text{ if } \alpha = f \wedge \forall i \leq m, \varepsilon_i \in L(E_i), \\
d_\alpha(\mathrm{op}(E_1, \ldots, E_k)) &= \mathrm{op}(d_\alpha(E_1), \ldots, d_\alpha(E_k)), \\
d_\alpha(E_1 \cdot_b E_2) &= \begin{cases} (d_b(E_1) \cdot_b E_2) \circ_1 d_b(E_2) & \text{if } \alpha = b, \\ d_\alpha(E_1) \cdot_b E_2 + (d_b(E_1) \cdot_b E_2) \circ_1 d_\alpha(E_2) & \text{if } \alpha \in \Sigma_0 \setminus \{b\}, \\ d_\alpha(E_1) \cdot_b E_2 & \text{otherwise,} \end{cases}
\end{aligned}$$

$$d_\alpha(E \circ (E_1, \dots, E_k)) = \sum_{1 \leq j \leq k} E \circ ((\underline{E_l})_{1 \leq l \leq j}, d_\alpha(E_j), (\underline{E_l})_{j+1 \leq l \leq k}) + \begin{cases} d_{\alpha((\varepsilon_{j p_l})_{1 \leq l \leq n})}(E) \circ (\varepsilon_1, (\underline{E_l})_{1 \leq l \leq k | \forall z, l \neq p_z}) \\ \quad \text{if } \forall 1 \leq l \leq n, \exists 1 \leq p_l \leq k, \varepsilon_l \in L(E_{p_l}) \\ \varnothing_{\mathrm{Ind}_\varepsilon(E \circ (E_1, \dots, E_k)) \setminus \{1, \dots, n\}} \quad \text{otherwise,} \end{cases}$$

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

$$d_\alpha(E^{*_b}) = \begin{cases} (d_b(E))^{\circledast} \cdot_b E^{*_b} & \text{if } \alpha = b, \\ ((d_b(E))^{\circledast} \circ (d_\alpha(E))) \cdot_b E^{*_b} & \text{otherwise,} \end{cases}$$

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 5.2.5. *Let* $t = f(t_1, \dots, t_n)$ *be a tree in* T_Σ *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*

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

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

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. As an example, consider the expression $E = \neg\varnothing_\varnothing$. When deriving E w.r.t. a nullary tree a, one must obtain an expression denoting all the trees that belongs to T_Σ in which one a was removed, that is the set of all the trees with only ε_1 as ε-indice. Applying the previously defined formulae:

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

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_Σ by removing to occurrences of a). Applying the previously defined formulae:

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

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

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

As a direct consequence of the inductive formulae of Section 2.2.1 and of Corollary 5.1.1, we get the following theorem.

Theorem 5.2.1. *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 (2.5) and Definition 5.2.3, it holds that

$$L(d_{\varepsilon_j}(E)) = \varepsilon_j^{-1}(L(E)).$$

2. Notice that the derivation formulae of Definition 5.2.4 are syntactical equivalents of the quotient formulae of Section 2.2.1 and of Corollary 5.1.1 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 composition product case. 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 have to be modified considering Equation (2.4). In the product case, the derivation of an expression by the tree $\alpha(\varepsilon_{j_1}, \ldots, \varepsilon_{j_n})$ has to be considered. However, by considering this particular case in the induction, one can check that

$$\begin{aligned} L(d_{\alpha(\varepsilon_{j_1},\ldots,\varepsilon_{j_n})}(E)) &= L(d_\alpha(d_{\varepsilon_{j_1+n-1}}(\cdots d_{\varepsilon_{j_{n-1}+1}}(d_{\varepsilon_{j_n}}(E))\cdots))) \\ &= \alpha^{-1}(d_{\varepsilon_{j_1+n-1}}(\cdots d_{\varepsilon_{j_{n-1}+1}}(d_{\varepsilon_{j_n}}(E))\cdots)), \end{aligned}$$

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 $\alpha(\varepsilon_{j_1}, \ldots, \varepsilon_{j_n})^{-1}(L(E))$ from Equation (2.8). We can conclude following Equation (2.10).

3. Finally, according to Equation (2.8) and item 2, Definition 5.2.5 implies that

$$L(d_t(E)) = t^{-1}(L(E)).$$

□

Example 5.2.1. *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

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

Then:

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

5.3 Tree Automaton Construction

We have seen in Chapter 3 how to construct a non deterministic tree automaton from a regular expression using the notion of derivation but in a Top-Down interpretation. This method is due to Kuske and Mienske [42] which is inspired from the Antimirov construction [2] in the case of words.

In 2017, Champarnaud *et al.* [19] have extended the inductive formulas of quotients to tree languages, following the Bottom-Up interpretation.

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

Definition 5.3.1. *Given a tree expression E over an alphabet Σ, let us see how we compute the tree automaton from a valid tree expression:*

- *First, we compute the tree automaton $A_0 = (\Sigma, D_0(E), F_0, \delta_0)$ where*

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

- *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, we define the tree automaton $A_n = (\Sigma, D_n(E), F_n, \delta_n)$.*
  - *We consider the transition set δ_n inductively defined by*

$$\begin{aligned} \delta_n = \{((E'_1, \ldots, E'_m), f, d_t(E)) \mid & t = f(\mathrm{tree}_{n-1}(E'_1), \ldots, \mathrm{tree}_{n-1}(E'_m)), \\ & f \in \Sigma_m, \\ & E'_1, \ldots, E'_m \in D_{n-1}(E)\}. \end{aligned} \tag{5.3}$$

  - *Next, we 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*

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

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 $f(\mathrm{tree}_{n-1}(E'_1), \ldots, \mathrm{tree}_{n-1}(E'_k))$.

– *Last, we consider the set*

$$F_n = \{E' \in D_n(E) \mid \varepsilon_1 \in L(E')\}, \tag{5.4}$$

- *Finally, let $A(E)$ be the fixed point, if it exists, of this process, called the* Bottom-Up derivative tree automaton *of E.*

This construction leads to a deterministic tree automaton and the validity of this construction does not depend on the choice of the tree_n functions.

Theorem 5.3.1. *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 the three ACI rules (associativity, commutativity and idempotence of the sum) are sufficient to obtain a finite tree automaton, isomorphic to the classical Brzozowski's automaton (for words).

Example 5.3.1. *Let us consider the tree expressions defined in Example 5.2.1,* 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 by applying the instructions of Definition 5.3.1.

- *First of all, let us compute the tree automaton $A_0 = (\Sigma, D_0(E), F_0, \delta_0)$:*

$$D_0(E) = \{d_a(E), d_b(E), d_c(E)\}.$$

Hence,

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

The tree automaton A_0 is represented in Figure 5.1.

FIGURE 5.1: The Tree Automaton A_0.

Now, we choose a function to define tree_0*:*

$$\text{tree}_0(d_a(E)) = a, \qquad \text{tree}_0(d_b(E)) = b.$$

- *After that, we show how to compute the tree automaton* $A_1 = (\Sigma, D_1(E), F_1, \delta_1)$.

  According to Equation 5.3, it is sufficient to compute the derivatives of E *w.r.t. the trees in the set* $\{f(a,a), f(a,b), f(b,a), f(b,b), g(a), g(b)\}$:

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

  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 5.2.*

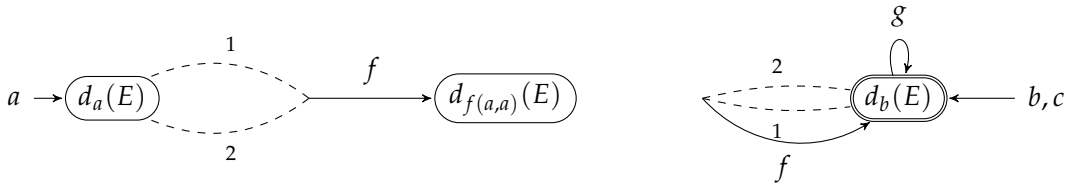


FIGURE 5.2: The Tree Automaton A_1.

- *The tree automaton* A_2 *can be computed thorough 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 5.3.*

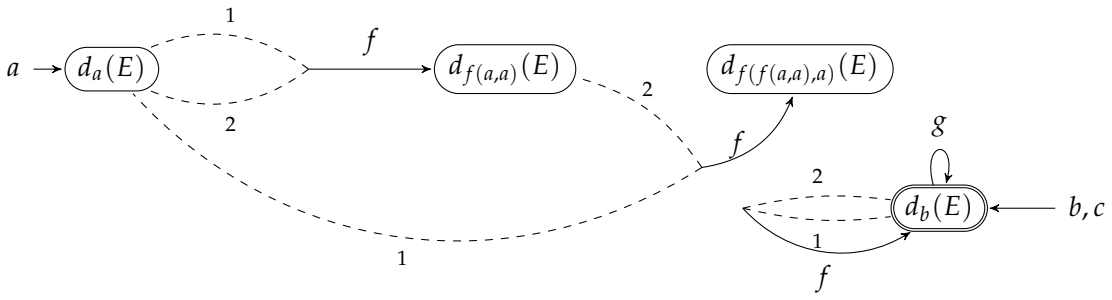


FIGURE 5.3: The Tree Automaton A_2.

- *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 new computations:*

$$d_{f(t,t)}(E) = d_{f(t,b)}(E) = d_{f(b,t)}(E) = d_b(E), \quad d_{g(t)}(E) = d_t(E).$$

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 6.6.

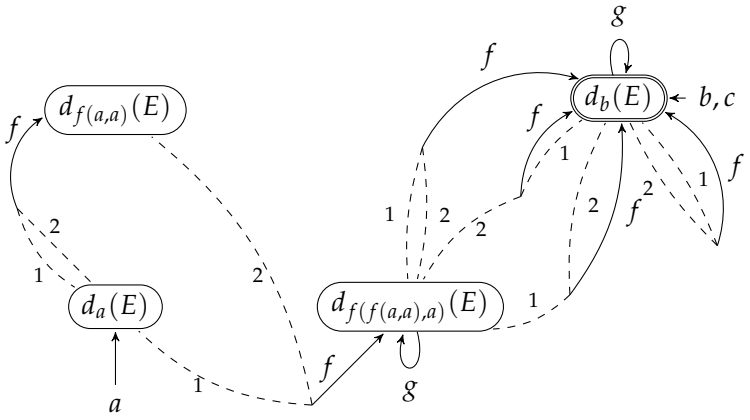


FIGURE 5.4: The Bottom-Up Derivative Tree Automaton of E.

5.4 Conclusion

We have shown in this chapter, how to compute the Bottom-Up derivative tree automaton from an extended regular expression as a fixed point of an inductive construction when it exists. Before that, we integrate the Boolean homogeneous operations in a tree expression in order to define the extended tree expression. Then, we made sure that the language denoted by the extended tree expression is homogeneous by introducing the notion of valid tree expression. Finally, in order to guarantee that the set of states (the set of derivatives) of the the constructed tree automaton is finite, we applied the well-known ACI rules of the sum. These rules are the same as used in the case of words, the question resides on the sufficiency of these rules to prove that the set of derivatives is finite. Seen that we deal with symbols of rank greater or equals to one and with operations like the composition $\circ$, the a-product, the iterated composition $^{\circledast}$ which are somehow "complicated". However, all the examples that we have made (taking into account the ACI rules) halt. So for now, this is only an hypothesis to justify maybe in our future works.

With this chapter, we conclude the first part of our thesis. The second part will be presented in the next chapter, we will focus on the essential element of the first part: *the ranked trees*. We will suggest formulae to count the number of ranked trees over a finite set of symbols by size and by height using combinatorial approaches.

Part II

Enumeration of Trees

6 Enumeration of Some Ranked Trees

The structure of trees is widely studied and used whether in the field of computer science [60, 39, 40] or in mathematics especially in graph theory and combinatorics [64].

Counting was the interest of a lot of mathematicians. Cayley [16] gave the enumeration of labeled trees of n nodes *i.e* the trees where each internal node has assigned unique number from 1 to n. It is shown in [64] that the number of binary trees of n vertices is *the Catalan numbers*. These latter were appeared to solve numerous combinatorial problems see [56, 58] and they are given by $C_n = \frac{1}{n+1}\binom{2n}{n}$ (Sloane's A000108) where $\binom{2n}{n}$ denotes a binomial coefficient.

In this chapter, we aim to give an enumeration of *ranked trees* [23] using combinatorial methods. In Section 6.1 we recall Catalan numbers and their generalizations. In Sections 6.2, 6.3 and 6.4, we show how to compute the number of ranked trees over a special kind of set of symbols by size and height as used in the case of rooted trees [53] over a particular set of symbols.

6.1 Catalan Numbers and Generalizations

For all $n, k \in \mathbb{Z}^+$ satisfying $k \leq n$ we denote by $\binom{n}{k}$ the binomial coefficient defined for $0 \leq k \leq n$ by

$$\binom{n}{k} = \frac{n!}{k!(n-k)!},$$

with the convention $\binom{n}{k} = 0$ for $k > n$ or $k < 0$.

For all $n_1, \ldots, n_r \in \mathbb{Z}$ and $n \in \mathbb{Z}^+$ we denote by $\binom{n}{n_1,\ldots,n_r}$ the multinomial coefficient defined for $n_1 + \cdots + n_r = n$ by

$$\binom{n}{n_1, \ldots, n_r} = \frac{n!}{n_1! \cdots n_r!},$$

and $\binom{n}{n_1,\ldots,n_r} = 0$ for $n_i < 0$ $(1 \leq i \leq r)$ or $n_1 + \cdots + n_r \neq n$.

One of the most advantage of this coefficient is the permission to omit the condition of the sum, that means, instead of writing $\sum_{\substack{n_1,\ldots,n_r \\ n_1+\cdots+n_r=n}} \binom{n}{n_1,\ldots,n_r}$ we can write directly $\sum_{n_1,\ldots,n_r} \binom{n}{n_1,\ldots,n_r}$.

The Catalan numbers $(C_m)_{m\geq 0}$ were appeared to solve numerous combinatorial problems (see [58]) and they are given by $C_m = \frac{1}{m+1}\binom{2m}{m}$ (Sloane's A000108) where $\binom{2m}{m}$ denotes the central binomial coefficient.

The k-Catalan numbers (see [57] and references therein) are the generalization of the Catalan numbers denoted by $(C_m^k)_{m\geq 0}$ for $k \in \mathbb{N}$.

A lot of mathematicians gave quite many combinatorial interpretations of these numbers [31, 32, 36, 41] and they are defined by

$$C_m^k = \frac{1}{km+1}\binom{km+1}{m} = \frac{1}{(k-1)m+1}\binom{km}{m}.$$

There exists a family of these numbers called *Fuss Catalan numbers* which also solve enumeration problems [25, 41, 7, 24, 55] and they are given for $n, r, p \in \mathbb{N} \setminus \{0\}$ by

$$C_m(k_1, k_2) = \frac{k_1}{mk_2 + k_1}\binom{mk_2 + k_1}{m} = \frac{k_1}{(k_2 - 1)m + k_1}\binom{k_2 m + k_1 - 1}{m}.$$

Notice that when $k = 2$, we get the classical Catalan numbers, C_m.

Theorem 6.1.1 (Stanley [57]). *A sequence $U_m(k)$ is a k-Catalan sequence if and only if $U_m(k) = \sum_{i_1+\cdots+i_k=m-1} U_{i_1}(k) \dots U_{i_k}(k)$.*

6.2 Enumeration of Trees over $\Sigma = \{f^k, \alpha_1, \dots, \alpha_s\}$ by Size

In this section, we give formulas to calculate the number of trees by size over $\Sigma = \{f^k, \alpha_1, \dots, \alpha_s\}$. This set contains s symbols of rank zero where $s \geq 1$ and m symbols of f of rank $k \geq 1$. One can wonder, given a finite set of symbols Σ. Is it possible to build trees using all these symbols, at least once, with any given size?

Proposition 6.2.1. *Let $\Sigma = \{f^k, \alpha_1, \dots, \alpha_s\}$. Let n be the size of a given tree. Then n satisfies*

$$n = mk + 1, \tag{6.1}$$

where m is the number of occurrences of f in the tree and the α_i's ($1 \leq i \leq s$) are placed on the leaves of the tree.

Proof. The proof is given by induction over the number of appearance of the symbol f.

- If f appears 'once', in the tree, then it produces k sons. These sons contain all the symbols of rank zero α_i where $i \in \{1, \dots, s\}$ that appear at least once which means s should be less or equal to k ($k \geq s$). Thus, we obtain at least $k + 1$ nodes in total.

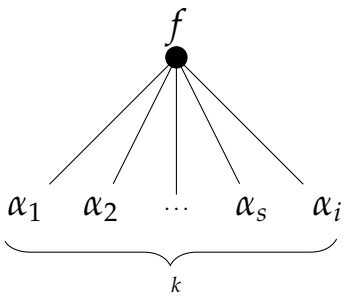


FIGURE 6.1: Tree with $k + 1$ nodes.

- If f appears 'twice', in the tree, then they produce $2k$ sons and $2k - 1$ leaves. These leaves must be greater or equal to s ($2k - 1 \geq s$). As a direct consequence, we obtain at least $2k + 1$ nodes in total.

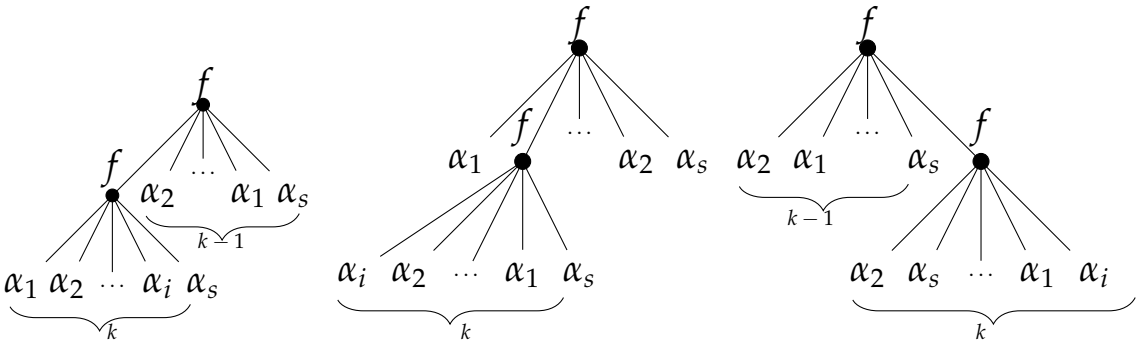
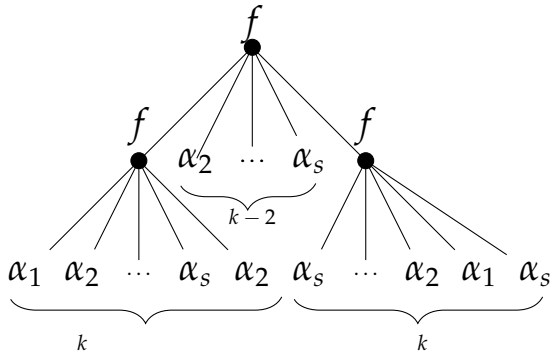


FIGURE 6.2: Trees with $2k+1$ nodes.

- If f appears 'three times', in the tree, then they produce $3k$ sons and $3k-2$ leaves which must be greater or equal to s ($3k-2 \geq s$). Therefore, at least $3k+1$ nodes will be produced in total. Two possibilities have to be considered:

  1. Either both of f are in the same level in the tree, we just have to permute the location of these two f's as follows:



  2. Or, the third f will be placed in one of k's sons of the second f, as follows:

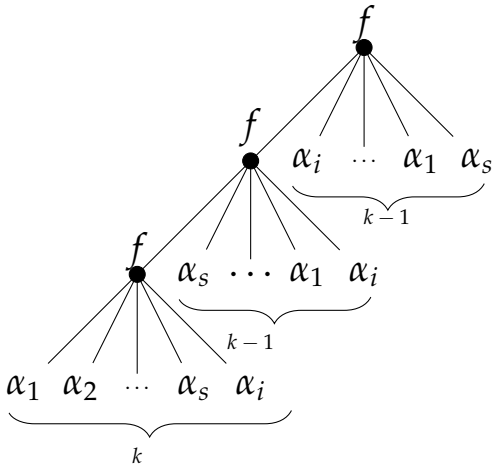


FIGURE 6.3: Trees with $3k+1$ nodes.

- If f appears m times, in the tree, then mk sons will be produced with $mk-(m-1) \geq s$ leaves. Consequently, we obtain $mk+1$ nodes in total.

Let us suppose that the hypothesis is true for m, is it true for $m+1$? The number of nodes of $m+1$ appearances of f is the number of nodes when f appears m times plus the number of nodes produced by adding one f in the tree. That means $(mk+1)+k$ nodes in total. As a conclusion, for $m+1$ occurrences of f, we have $n=(m+1)k+1$. □

Notice that for a fixed m and k, we obtain a unique number n as a size of the tree and *vice versa*: for a fixed n there is a unique number m of occurrences of the symbol f.

Proposition 6.2.2. *Let $\Sigma=\{f^k,\alpha_1,\ldots,\alpha_s\}$. Let m be the number of occurrences of f in the tree. Then m satisfies*

$$m=\frac{n-1}{k}, \tag{6.2}$$

where n is the size of a given tree and the α_i's ($1\leq i\leq s$) are placed on the leaves of the tree.

The following theorem shows how to count the number of trees over the set of symbols Σ considering a given size (according to Proposition 6.2.1).

Theorem 6.2.1. *Let $k,s\in\mathbb{Z}^+$, $n,m\in\mathbb{N}$, $\Sigma=\{f^k,\alpha_1,\ldots,\alpha_s\}$ and let $T_n(k,m;\alpha_1,\ldots,\alpha_s)$ be the number of ranked trees over Σ of size n with m is the number of occurrences of f and $\alpha_1,\ldots,\alpha_s$ are symbols of rank zero. Then*

$$\begin{aligned}T_n(k,m;\alpha_1,\ldots,\alpha_s)&=C_m^k\sum_{n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1}\binom{n-m}{n_{\alpha_1},\ldots,n_{\alpha_s}}\\&=C_m^k\sum_{n_{\alpha_1},\ldots,n_{\alpha_s}\geq 0}\binom{n-m}{n_{\alpha_1}+1,\ldots,n_{\alpha_s}+1}.\end{aligned}$$

Proof. The proof is given into two steps:

1. Calculate the number of ways to place the internal nodes f in the tree. Let $U_m(k)$ be the number of ways to place m times the symbol f of arity k in the tree.
   - for $m=0$, $U_0(k)=1$.
   - for $m=1$, in this case, we should have only one symbol f to place it in the tree. This symbol will be placed on the root and we only obtain one structure. Therefore, $U_1(k)=1$.

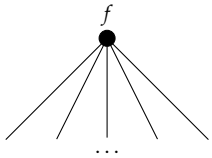


FIGURE 6.4: An illustration of the structure of the tree when $m=1$.

   - for $m>1$: we place "one" of f on the root of a given tree, it remains $m-1$ of f in order to place them, we distinguish these possibilities:
     - Place all the $m-1$ occurrences of f's ($U_{m-1}(k)$) in the internal nodes of the subtree colored in Figure 6.5.

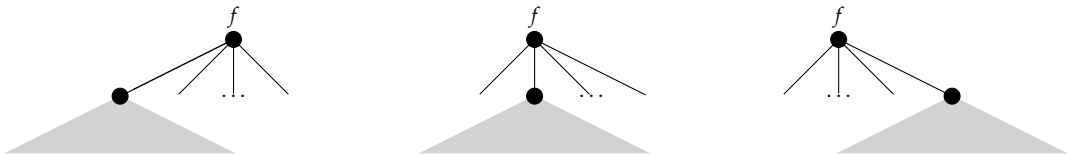


FIGURE 6.5: Structure of trees when $m > 1$.

– Place 'one time' f on root's sons and the $m-2$ remaining symbols will be placed on the subtree colored in Figure 6.6.

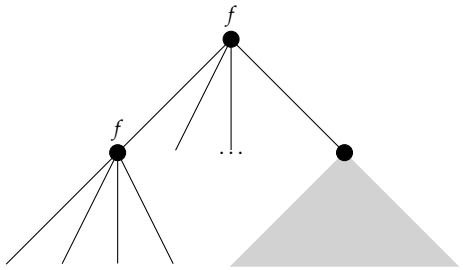


FIGURE 6.6: Another structure of trees when $m > 1$.

– Place 'two times' f on the root's sons and the $m-3$ remaining symbols will be placed on the subtree colored in Figure 6.7.

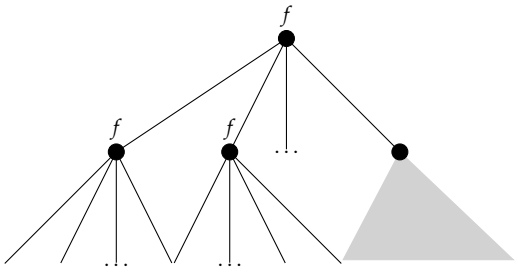


FIGURE 6.7: Another structure of trees when $m > 1$.

It satisfies

$$U_m(k) = \underbrace{U_0(k)U_0(k)\cdots U_0(k)U_{m-1}(k)}_{k \text{ terms}} + \underbrace{U_1(k)U_0(k)\cdots U_0(k)U_{m-2}(k)}_{k \text{ terms}} + \underbrace{U_1(k)U_1(k)U_0(k)\cdots U_0(k)U_{m-3}(k)}_{k \text{ terms}} + \cdots + \underbrace{U_{m-1}(k)U_0(k)\cdots U_0(k)}_{k \text{ terms}},$$

for $m \geq 2$

$$U_m(k) = \sum_{i_1+\cdots+i_k=m-1} U_{i_1}(k)U_{i_2}(k)\cdots U_{i_k}(k).$$

By Theorem 6.1.1

$$U_m(k) = C_m^k. \tag{6.3}$$

2. Calculate the number of ways to place $\alpha_1, \ldots, \alpha_s$ in the leaves of a tree. Given n the size of a tree t and m the number of occurrences of f in t. The number of leaves of t is $n-m$. So how one can place all the α_i's for $i = 1, \ldots, s$ at least once in the leaves. This

is equivalent to count the number of ways to put n_{α_1} (which represents the number of appearance of α_1 in t) from $n-m$ leaves then to put n_{α_2} of α_2 from $n-m-n_{\alpha_1}$ leaves,..., then to put $n_{\alpha_{n_{s-1}}}$ in $n-m-n_{\alpha_1}-\cdots-n_{\alpha_{s-2}} = n_{\alpha_{s-1}}+n_{\alpha_s}$ leaves and last to put n_{α_s} in the remaining leaves which is n_{α_s}. Formally,

$$\sum_{n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1} \binom{n-m}{n_{\alpha_1}} \binom{n-m-n_{\alpha_1}}{n_{\alpha_2}} \cdots \binom{n_{\alpha_{s-1}}+n_{\alpha_s}}{n_{\alpha_{s-1}}} \binom{n_{\alpha_s}}{n_{\alpha_s}},$$

which is equal to

$$\sum_{n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1} \binom{n-m}{n_{\alpha_1}, n_{\alpha_2}, \ldots, n_{\alpha_s}}.$$

As a conclusion, the number of trees $T_n(k,m;\alpha_1,\ldots,\alpha_s)$ over $\Sigma = \{f^k,\alpha_1,\ldots,\alpha_s\}$ is

$$T_n(k,m;\alpha_1,\ldots,\alpha_s) = C_m^k \sum_{n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1} \binom{n-m}{n_{\alpha_1}, n_{\alpha_2}, \ldots, n_{\alpha_s}}.$$

□

C_m^k \ m	0	1	2	3	4	5	Formula	Sloane
C_m^2	1	1	2	5	14	42	$\binom{2m}{m}\frac{1}{m+1}$	A000108
C_m^3	1	1	3	12	55	273	$\binom{3m}{m}\frac{1}{2m+1}$	A001764
C_m^4	1	1	4	22	140	969	$\binom{4m}{m}\frac{1}{3m+1}$	A002293
C_m^5	1	1	5	35	285	2530	$\binom{5m}{m}\frac{1}{4m+1}$	A002294
C_m^6	1	1	6	51	506	5481	$\binom{6m}{m}\frac{1}{5m+1}$	A002295
C_m^7	1	1	7	70	819	10472	$\binom{7m}{m}\frac{1}{6m+1}$	A002296
C_m^8	1	1	8	92	1240	18278	$\binom{8m}{m}\frac{1}{7m+1}$	A007556

TABLE 6.1: Some values of the number of ways to place m times the symbol f of rank k in the internal nodes of a tree.

Example 6.2.1. *Let us consider $\Sigma = \{f^3, a, b, c\}$. What is the number of trees that could be built from Σ?*

- *First of all, the size n of the trees t should satisfy, by Proposition 6.2.1, the fact that $n = 3m+1$. Thus, $n = \{4, 7, 10, 13, \ldots\}$. Let us calculate the number of trees over Σ when the size of a tree is 10, in this case the number of occurrences 'm' of f is 3.*
- *Next, we compute the number of ways to partition the internal nodes f in the tree. Let C_3^3 be the number of ways to place 'three' times f of arity 3 in the tree of size 10. According to Equation (6.3), we get $C_3^3 = \frac{1}{7}\binom{9}{3} = 12$.*

- *Last, the number of ways of placing* 3 *times* f *in a tree of size* 10 *is* 12 *trees, and they are represented in Figure 6.8 as follows:*

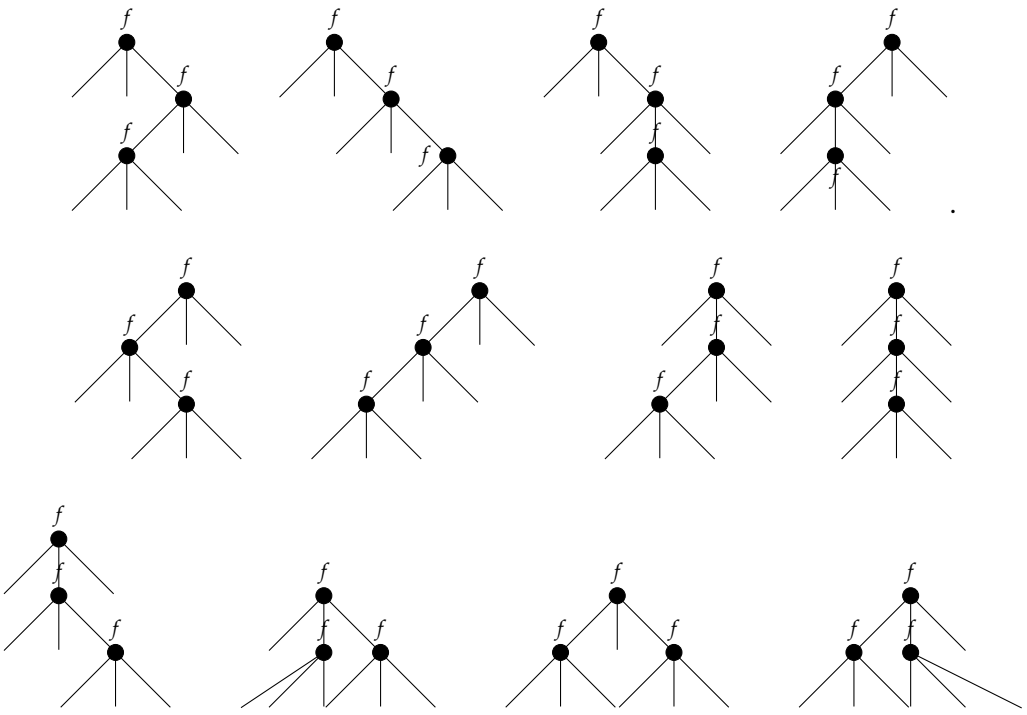


FIGURE 6.8: Number of ways of placing the internal nodes f in a tree of size 10.

- *In the next step, we calculate the number of ways of partitioning the symbols of rank zero* a, b *and* c *in* 7 *leaves of a tree of size* 10*. To do so, we use a combinatorial approach as follows: let* n_a, n_b *and* n_c *be the number of appearance of the symbol* a*,* b *and* c *respectively in the leaves of the tree. Partitioning* 7 *symbols of rank zero into* 3 *subsets such that each subset contains* $n_a \geq 1$*,* $n_b \geq 1$ *and* $n_c \geq 1$ *symbols is equivalent to*

$$\sum_{n_a,n_b,n_c \geq 1} \binom{7}{n_a, n_b, n_c} = 1806.$$

- *Hence, we have* 1806 *ways to place the symbols* a, b *and* c *at least once in the leaves of a tree of size* 10*.*

- *Finally, by Theorem 6.2.1, the total number of trees* $T_{10}(3, 3; a, b, c)$ *over* Σ *is*

$$T_{10}(3, 3; a, b, c) = 12 \times 1806 = 21672.$$

6.3 Enumeration of Trees over Σ by Size and by Height

We will give in this section a formula to calculate the number of trees over Σ by considering the height h of trees which is the number of edges on the longest path between the root and the leaf.

Theorem 6.3.1. *Let* $T_n(k, m, h; \alpha_1, \ldots, \alpha_s)$ *be the number of trees over* Σ *of height* h *and size* n *where* $n = mk + 1$*,* m *is the number of occurrences of* f *and* $\alpha_1, \ldots, \alpha_s$ *are symbols of arity zero in the tree.*

Let $U_{m,h}(k)$ be the number of ways to place m times f in the internal nodes of a tree of height h.

$$T_n(k,m,h;\alpha_1,\ldots,\alpha_s) = U_{m,h}(k) \sum_{n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1} \binom{n-m}{n_{\alpha_1},\ldots,n_{\alpha_s}}, \tag{6.4}$$

where

$$U_{m,h}(k) = \begin{cases} 0 & \text{for } h > m, \\ k^{m-1} & \text{for } h = m, \\ 2(h-2)k^{m-3}\binom{k}{2} + k^{m-3}\binom{k}{2} & \text{for } h = m-1. \end{cases} \tag{6.5}$$

Notice that for the case $h \leq m-2$, we have to treat each case apart, we would have to analyze the structures of trees and find a formula to each rank which seems "difficult for the moment".

Proof. Given a tree t of height h and size n built from Σ. Let us calculate the number of ways to place m occurrences of f in the internal nodes of t of height h. To do so, we consider these cases:

- The case where $h > m$: in this case the size of a given tree will not be respected and we can not construct such a tree.
- The case where $h = m$: let us place the first f in the root of the tree t, it remains $m-1$ occurrences of f to place them in a tree of height m. The idea is to place in each level only one occurrence of f. Each placed symbol has k possibilities to place it. Thus, we have k^{m-1} trees over Σ. The only structure of this case is shown in Figure 6.9.

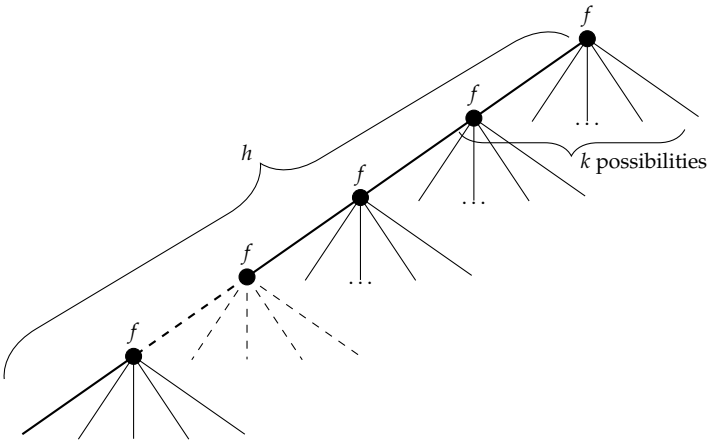
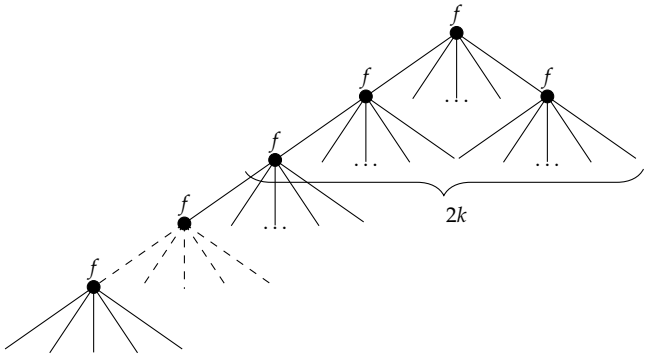
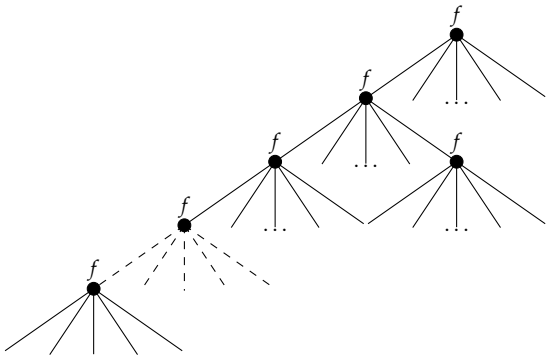


FIGURE 6.9: The structure of trees when $h = m$.

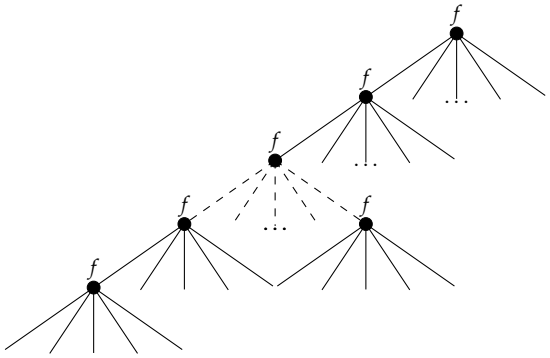
- The case where $h = m-1$: in such structure there exist h levels. The idea is to place exactly one time f in $h-2$ levels, the remaining level contains exactly 2 occurrences of f. The allowed structures are presented in the figure below.



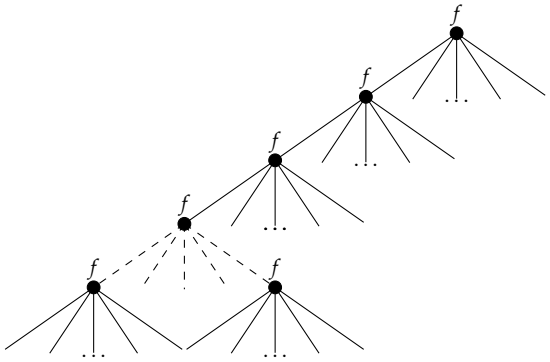
(a)



(b)



(c)



(d)

FIGURE 6.10: The structure of trees for $h = m - 1$.

Notice that there exist $\binom{k}{2}$ possibilities to place the two symbols of f differently in $h-2$ levels (see structures (a), (b) and (c) of Figure 6.10). Moreover, the successor level of these levels contains only one appearance of f which will be placed in $2k$ possibilities. The $m-4$ remaining symbols of f would have k possibilities to place them. Therefore, we get

$$2k(h-2)\binom{k}{2}k^{m-4} = 2(h-2)\binom{k}{2}k^{m-3}.$$

Furthermore, the only case where the two appearances of the symbol f are situated in the last level (see structure (d) of Figure 6.10) we would have $\binom{k}{2}$ possibilities to place them. The $m-3$ remaining symbols will be placed in k different ways. That means

$$\binom{k}{2}k^{m-3}.$$

Therefore, the number of ways to place m occurrences of f in a tree of height h where $h = m-1$ is

$$2(h-2)\binom{k}{2}k^{m-3} + \binom{k}{2}k^{m-3}.$$

Last, we calculate the number of ways to place $\alpha_1, \ldots, \alpha_s$ in the leaves of a tree. Given n the size of a tree t and m the number of occurrences of f in t. The number of leaves of t is $n-m$. So how one can place all the α_i's for $i = 1, \ldots, s$ at least once in the leaves. This is equivalent to count the number of ways to put n_{α_1} (which represents the number of appearance of α_1 in t) from $n-m$ leaves then to put n_{α_2} of α_2 from $n-m-n_{\alpha_1}$ leaves,..., then to put $n_{\alpha_{n_{s-1}}}$ in $n-m-n_{\alpha_1}-\cdots-n_{\alpha_{s-2}} = n_{\alpha_{s-1}}+n_{\alpha_s}$ leaves and last to put n_{α_s} in the remaining leaves which is n_{α_s}. Formally,

$$\sum_{\substack{n_{\alpha_1}+\cdots+n_{\alpha_s}=n-m \\ n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1}} \binom{n-m}{n_{\alpha_1}}\binom{n-m-n_{\alpha_1}}{n_{\alpha_2}}\cdots\binom{n_{\alpha_{s-1}}+n_{\alpha_s}}{n_{\alpha_{s-1}}}\binom{n_{\alpha_s}}{n_{\alpha_s}},$$

which is equal to

$$\sum_{n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1} \binom{n-m}{n_{\alpha_1},\ldots,n_{\alpha_s}}.$$

As a conclusion, the number of trees $T_n(k,m,h;\alpha_1,\ldots,\alpha_s)$ over $\Sigma = \{f^k,\alpha_1,\ldots,\alpha_s\}$ is

$$T_n(k,m,h;\alpha_1,\ldots,\alpha_s) = U_{m,h}(k)\sum_{n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1}\binom{n-m}{n_{\alpha_1},\ldots,n_{\alpha_s}},$$

□

Bellow, we give some values of the number of ways to place f in the internal nodes of trees of size n and height h.

m	n \ h	1	2	3	4	5	6	7	
1	3	1							1
2	5		2						2
3	7		1	4					5
4	9			6	8				14
5	11			6	20	16			42
6	13			4	40	56	32		132
7	15			1	68	152	144	64	429

TABLE 6.2: Number of ways to place f of rank 2 in a tree by size n and height h.

m	n \ h	1	2	3	4	5	6	7	
1	4	1							1
2	7		3						3
3	10		3	9					12
4	13		1	27	27				55
5	16			57	135	81			273
6	19			96	522	567	243		1428

TABLE 6.3: The number of ways to place f of rank three in a tree by size n and height h.

m	n \ h	1	2	3	4	5	6	7	
1	5	1							1
2	9		4						4
3	13		6	16					22
4	17		4	72	64				140
5	21		1	232	480	256			969
6	25			20	3352	2688	1024		7084

TABLE 6.4: The number of ways to place f of rank four in a tree by size n and height h.

Notice that the number of ways to place f in the internal nodes of a tree t of size n (which is the k-Catalan) is the computations of the horizontal sums of the trees of size n and height $h \geq \lfloor log_k((k-1)n) \rfloor$ (See [59, p. 237]). As an example, the number of ways to place f in a tree of size 13 is 132 which is also the sum of the trees of height 3, of height 4, of height 5 and of height 6 which are respectively 4, 40, 56 and 32 (see Table 6.2). In the other hand, the number of trees of height 3 and size ≤ 15 is the sum of 4 trees of size 7, 6 trees of size 9, 6 trees of size 11, 4 trees of size 13 and last 1 tree of size 15.

Example 6.3.1. *Let us consider the set of symbols of Example 6.2.1 defined by . Let us calculate the number of trees of size* 10, $m = 3$ *and height* h *where* $2 \leq h \leq 3$.

First, we calculate the number of ways to place the symbol f *of rank* 3 *in the internal nodes of trees.*

- *The case when the height of the trees is* 2. *By Equation 6.5 case* $h = m - 1$

$$U_{m,h}(k) = 2(h-2)k^{m-3}\binom{k}{2} + k^{m-3}\binom{k}{2}$$
$$U_{3,2}(3) = \binom{3}{2} = 3,$$

and they are presented in the following figure.

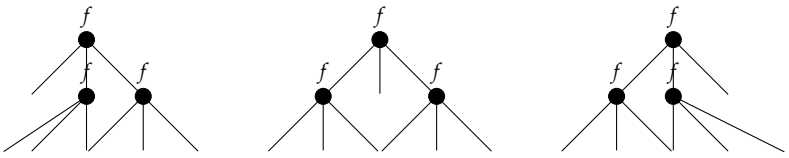


FIGURE 6.11: Number of ways of placing the internal nodes f in a tree of size 10 and height 2.

- *The case when the height of the trees is* 3*: By Equation 6.5 case* $h = m$

$$U_{m,h}(k) = k^{m-1}$$
$$U_{3,3}(3) = 3^2 = 9,$$

and they are presented in the following figure.

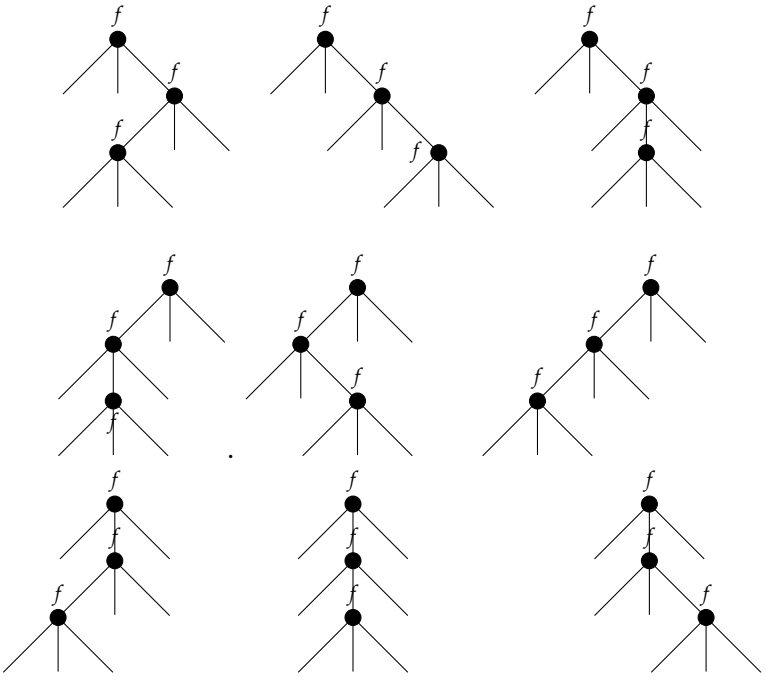


FIGURE 6.12: Number of ways of placing the internal nodes f in a tree of size 10 and height 3.

In the next step, we calculate the number of ways of partitioning the symbols of rank zero a, b *and* c *in* 7 *leaves of a tree of size* 10*. To do so, we use a combinatorial approach as follows: let* n_a, n_b *and* n_c *be*

the number of appearance of the symbol a, b and c respectively in the leaves of the tree. Partitioning 7 symbols of rank zero into 3 subsets such that each subset contains $n_a \geq 1$, $n_b \geq 1$ and $n_c \geq 1$ symbols is equivalent to

$$\sum_{n_a,n_b,n_c \geq 1} \binom{7}{n_a, n_b, n_c} = 1806.$$

- *Hence, we have* 1806 *ways to place the symbols a, b and c at least once in the leaves of a tree of size* 10.
- *Finally, by Theorem 6.3.1, the total number of trees $T_{10}(3,3,h;a,b,c)$ over Σ is*

$$T_{10}(3,3,h;a,b,c) = U_{3,h}(3) \times 1806.$$

6.4 Enumeration of Trees over $\Sigma = \{f_1^{k_1}, f_2^{k_2}, \alpha_1, \ldots, \alpha_s\}$ by Size

In this section, we count the number of trees over $\Sigma = \{f_1^{k_1}, f_2^{k_2}, \alpha_1, \ldots, \alpha_s\}$. This set contains s symbols of rank zero where $s \geq 1$ and two symbols of a different rank k_1 and k_2. We assume that the symbol $f_1^{k_1}$ appears 'once' in the root of a tree.

Proposition 6.4.1. *Let us consider the set of symbols Σ. Let n be the size of a tree. Then n should satisfy this condition:*

$$n = 1 + k_1 + mk_2. \tag{6.6}$$

Where m is the number of occurrences of f_2 in the tree.

The following Theorem counts the number of trees by size over Σ.

Theorem 6.4.1. *Let $\Sigma = \{f_1^{k_1}, f_2^{k_2}, \alpha_1, \ldots, \alpha_s\}$ be a set of symbols. Let $T_n(k_1, k_2, m; \alpha_1, \ldots, \alpha_s)$ be the number of ranked trees over Σ of size n.*

$$T_n(k_1,k_2,m;\alpha_1,\ldots,\alpha_s) = C_m(k_1,k_2) \sum_{\substack{n_{\alpha_1}+\cdots+n_{\alpha_s}=n-m-1 \\ n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1}} \binom{n-m-1}{n_{\alpha_1}, n_{\alpha_2}, \ldots, n_{\alpha_s}}, \tag{6.7}$$

where m is the number of occurrences of f_2 in the tree t and n, k_1, k_2 are non negative integers.

Proof. To prove the theorem, two steps have to be considered:

Step 1: Place m times f_2 in the internal nodes:

Let $U_{1,m}(k_1,k_2)$ be the number of ways to place one time f_1 of rank k_1 and m times f_2 of rank k_2 in a given tree t of size n.

Notice that $U_{0,m}(k_1,k_2) = U_m(k_2)$.

- If $m = 0$ then $U_{1,0}(k_1,k_2) = 1$.
- If $m = 1$ then we have k_1 possibilities to place f_2. Thus, $U_{1,1}(k_1,k_2) = k_1$.

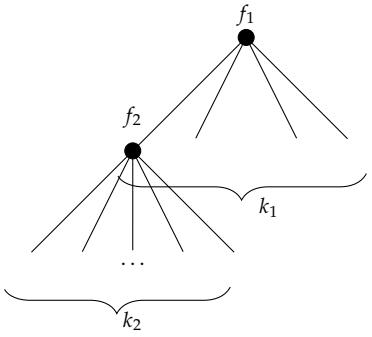


FIGURE 6.13: Illustration of the structure of trees for $m = 1$.

- If $m > 1$ then these possibilities are considered:
  - Place all the occurrences of f_2 on the subtree colored in Figure 6.14.

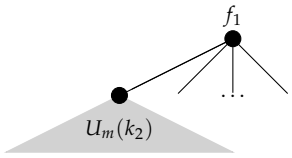


FIGURE 6.14: Structure of trees for $m > 1$.

  - Place one occurrence of f_2 on root's sons, the $m - 1$ remaining symbols of f_2 will be placed on the subtree colored on Figure 6.15.

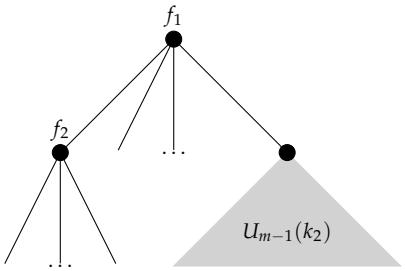


FIGURE 6.15: Another structure of trees for $m > 1$.

  - Place two of f_2 root's sons, the $m - 2$ remaining symbols of f_2 will be placed on the subtree colored in Figure 6.16.

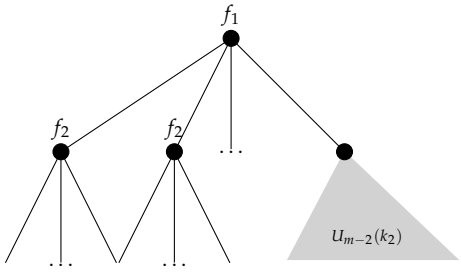


FIGURE 6.16: Another structure of trees for $m > 1$.

It can be defined by

$$U_{1,m}(k_1,k_2) = U_0(k_2)U_0(k_2)\cdots U_m(k_2) + U_1(k_2)U_0(k_2)\cdots U_{(m-1)}(k_2) + U_1(k_2)U_1(k_2)U_0(k_2)\cdots U_{(m-2)}(k_2) + \cdots + U_m(k_2)U_0(k_2)\cdots U_0(k_2).$$

$$U_{1,m}(k_1,k_2) = \begin{cases} 1 & \text{for } m = 0 \\ k_1 & \text{for } m = 1 \\ \sum\limits_{i_1+\cdots+i_{k_1}=m} U_{i_1}(k_2)U_{i_2}(k_2)\ldots U_{i_{k_1}}(k_2) & \text{otherwise.} \end{cases} \tag{6.8}$$

The values of this sequence represent the generalized Fuss-Catalan numbers.

$$U_{1,m}(k_1,k_2) = C_m(k_1,k_2) = \binom{k_2m+k_1-1}{m}\frac{k_1}{(k_2-1)m+k_1}. \tag{6.9}$$

Step 2: Place $\alpha_1,\ldots,\alpha_s$ in the leaves of t, and this is equivalent to count the number of ways to put n_{α_1} (which represents the number of appearance of α_1 in t) from $n-m-1$ leaves then to put n_{α_2} of α_2 from $n-m-1-n_{\alpha_1}$ leaves,..., then to put $n_{\alpha_{n_{s-1}}}$ in $n_{\alpha_{s-1}}+n_{\alpha_s}$ leaves and last to put n_{α_s} in the remains leaves which is n_{α_s}. It can be seen as

$$\sum_{\substack{n_{\alpha_1}+\cdots+n_{\alpha_s}=n-m-1 \\ n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1}} \binom{n-m-1}{n_{\alpha_1}}\binom{n-m-1-n_{\alpha_1}}{n_{\alpha_2}}\cdots\binom{n_{\alpha_{s-1}}+n_{\alpha_s}}{n_{\alpha_{s-1}}}\binom{n_{\alpha_s}}{n_{\alpha_s}}.$$

Which is equivalent to

$$\sum_{\substack{n_{\alpha_1}+\cdots+n_{\alpha_s}=n-m-1 \\ n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1}} \binom{n-m-1}{n_{\alpha_1},n_{\alpha_2},\ldots,n_{\alpha_s}}.$$

As a conclusion, the number of trees $T_n(k_1,k_2,m;\alpha_1,\ldots,\alpha_s)$ over $\Sigma = \{f_1^{k_1}f_2^{k_2},\alpha_1,\ldots,\alpha_s\}$ is

$$T_n(k_1,k_2,m;\alpha_1,\ldots,\alpha_s) = C_m(k_1,k_2)\sum_{\substack{n_{\alpha_1}+\cdots+n_{\alpha_s}=n-m-1 \\ n_{\alpha_1},\ldots,n_{\alpha_s}\geq 1}} \binom{n-m-1}{n_{\alpha_1},n_{\alpha_2},\ldots,n_{\alpha_s}}.$$

□

Example 6.4.1. *Let us consider the set of symbols $\Sigma = \{f_1^3, f_2^4, a, b, c\}$. What is the number of trees that could be built from Σ?*

- *First of all, the size n of the trees t should, according to Proposition 6.4.1, be in $n = \{8, 12, 16, \ldots\}$.*
- *Let us calculate the number of trees over Σ when the size of a tree is* 12, i.e. *the number of occurrences of f_1 is once and f_2 is* 2.
- *We compute the number of ways to partition the internal nodes f_2 in the tree. To do so, let $C_2(3,4)$ be the number of ways to place f_1 of arity* 3 *and f_2 of arity* 4 *in the tree. According to Formula 6.9:*

$$C_2(k_1,k_2) = \binom{2k_2+k_1-1}{2}\frac{k_1}{2(k_2-1)+k_1}$$
$$C_2(3,4) = \binom{10}{2}\frac{1}{3} = 15.$$

So, the number of ways to place the internal nodes f_2 in the tree of size 12 *is* 15 *ways. The trees are presented in Figure 6.17.*

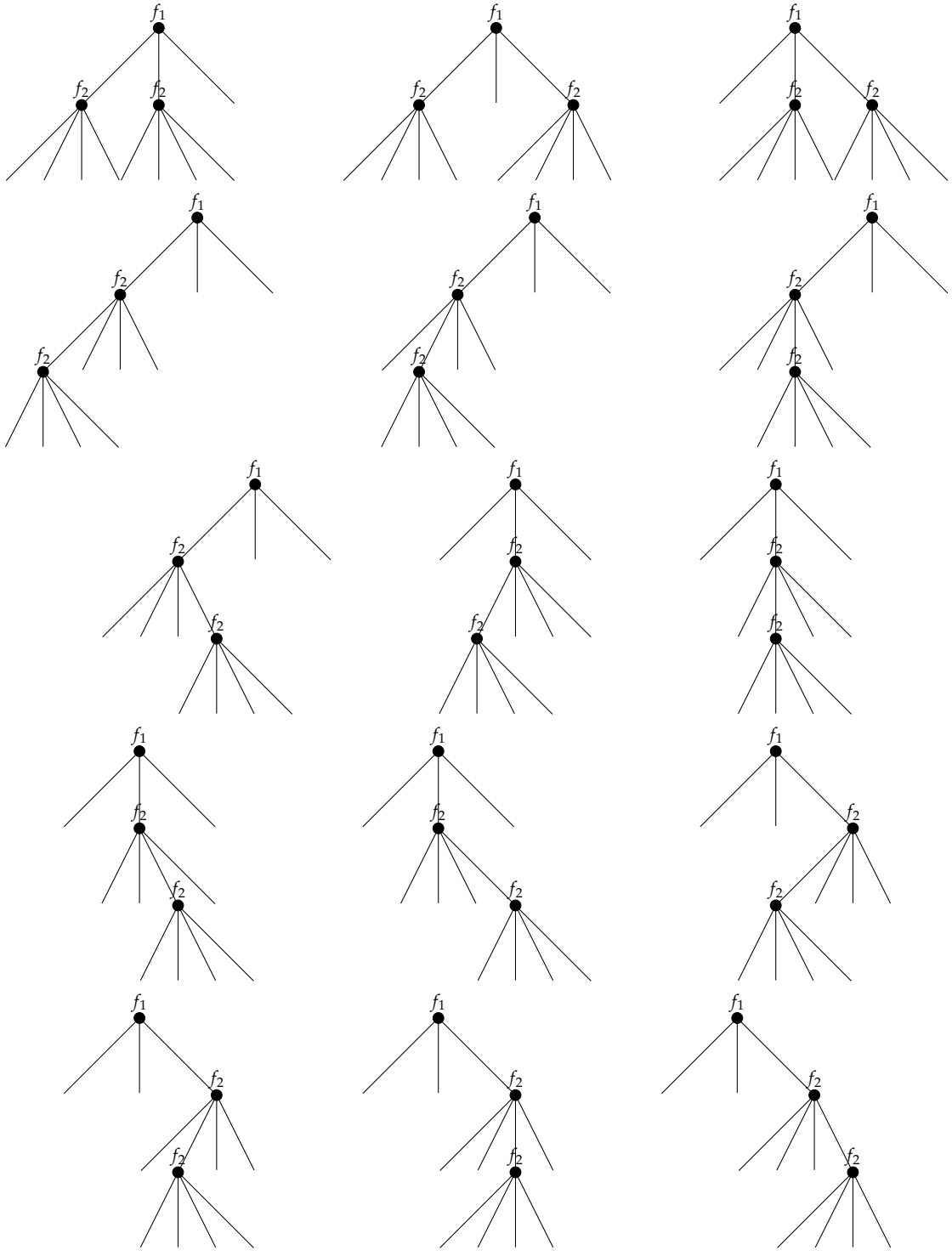


FIGURE 6.17: Number of ways of placing the internal nodes f_2 in a tree of size 12.

- *In the second step, we compute the number of ways to place a, b and c in the $n - m - 1 = 9$ leaves of the tree t.*

To do so, we use a combinatorial approach as follows: let $n_{\alpha_a}, n_{\alpha_b}$ *and* n_{α_c} *be the number of appearance of the symbol* a*,* b *and* c *respectively in* 9 *leaves. Partitioning* 9 *symbols of rank zero into* 3 *subsets such that each subset contains* $n_{\alpha_a} \geq 1$*,* $n_{\alpha_b} \geq 1$ *and* $n_{\alpha_c} \geq 1$ *symbols is equivalent to*

$$\sum_{\substack{n_{\alpha a}+n_{\alpha_b}+n_{\alpha c}=9 \\ n_{\alpha a},n_{\alpha_2},n_{\alpha c}\geq 1}} \binom{9}{n_{\alpha_a}, n_{\alpha_b}, n_{\alpha_c}} = 18150.$$

- *Hence, we have* 18150 *ways to place the symbols* a, b *and* c *at least once in the leaves of a tree of size* 12*.*
- *Finally, by Theorem 6.4.1, the total number of trees over* Σ *of size* 12 *is* $T_{12}(3, 4, 2; a, b, c) = 15 \times 18150$ *trees.*

$U_{1,m}(k_1,k_2)$ \ m	0	1	2	3	4	5	Formula	Sloane
$U_{1,m}(2,1)$	1	2	3	4	5	6	$\binom{m+1}{m}$	
$U_{1,m}(2,3)$	1	2	7	30	147	728	$\binom{3m+1}{m}\frac{2}{2m+2}$	A006013
$U_{1,m}(2,4)$	1	2	9	52	340	2394	$\binom{4m+1}{m}\frac{2}{3m+2}$	A069271
$U_{1,m}(2,5)$	1	2	11	80	665	5980	$\binom{5m+1}{m}\frac{2}{4m+2}$	A118969
$U_{1,m}(2,6)$	1	2	13	114	1150	12586	$\binom{6m+1}{m}\frac{2}{5m+2}$	A212071
$U_{1,m}(2,7)$	1	2	15	154	1827	23562	$\binom{7m+1}{m}\frac{2}{6m+2}$	A233832
$U_{1,m}(2,8)$	1	2	17	200	2728	40508	$\binom{8m+1}{m}\frac{2}{7m+2}$	A234461
$U_{1,m}(3,1)$	1	3	6	10	15	21	$\binom{m+2}{m}$	
$U_{1,m}(3,2)$	1	3	9	28	90	297	$\binom{2m+2}{m}\frac{3}{m+3}$	
$U_{1,m}(3,4)$	1	3	15	91	612	4389	$\binom{4m+2}{m}\frac{3}{3m+3}$	A006632
$U_{1,m}(3,5)$	1	3	18	136	1155	10530	$\binom{5m+2}{m}\frac{3}{4m+3}$	A118970
$U_{1,m}(3,6)$	1	3	21	190	1950	21576	$\binom{6m+2}{m}\frac{3}{5m+3}$	A212072
$U_{1,m}(3,7)$	1	3	24	253	3045	39627	$\binom{7m+2}{m}\frac{3}{6m+3}$	A233833
$U_{1,m}(3,8)$	1	3	27	325	4488	67158	$\binom{8m+2}{m}\frac{3}{7m+3}$	A234462
$U_{1,m}(4,1)$	1	4	10	20	35	56	$\binom{m+3}{m}$	
$U_{1,m}(4,2)$	1	4	14	48	165	572	$\binom{2m+3}{m}\frac{4}{m+4}$	A002057
$U_{1,m}(4,3)$	1	4	18	88	455	2448	$\binom{3m+3}{m}\frac{4}{2m+4}$	A006629
$U_{1,m}(4,5)$	1	4	26	204	1771	16380	$\binom{5m+3}{m}\frac{4}{3m+4}$	A118971
$U_{1,m}(4,6)$	1	4	30	280	2925	32736	$\binom{6m+3}{m}\frac{4}{5m+4}$	A212073
‘ $U_{1,m}(4,7)$	1	4	34	368	4495	59052	$\binom{7m+3}{m}\frac{4}{6m+4}$	
$U_{1,m}(4,8)$	1	4	38	468	6545	98728	$\binom{8m+3}{m}\frac{4}{7m+4}$	A234463

TABLE 6.5: Some values of the number of ways to place $f_1^{k_1}$ and $f_2^{k_2}$ in the internal nodes of a tree.

6.5 Conclusion

We have seen in this chapter, how to count the number of trees over two sets of symbols by size and by height by proposing formulas using combinatorial approaches.

7 Conclusion and Perspectives

Throughout this thesis, we have answered the sited questions in the introduction. In the first part, we prove Kleene theorem in the case of trees. We could propose constructive algorithms to convert a regular tree expression into a tree automaton, in a Bottom-Up way, recognizing the same language by extending the methods already suggested in the case of words.

A generalization from words to trees of the Glushkov's construction, the Ilie and Yu's method and the Brzozowski's algorithm of automata from regular expressions are presented. These constructions are based on a Bottom-Up interpretation of regular expressions, while previous constructions were based on a Top-Down interpretation.

First of all, we construct the position tree automata, the father tree automata and their compressed versions from a given regular expression in a Bottom-Up way.

The open questions left after this study are the following:

- Can we extend all the studies related to Glushkov automata in the case of words to the case of trees?
- Is it possible to compress the Top-Down position and follow tree automata in the same way as used in our study?
- As we have seen, the compression of transitions that we have used is based on the cartesian product, one can wonder if we can use a heuristic to reduce the number of transitions.
- Last, we have made a comparative study in point of view the number of transitions between Top-Down position automaton, Bottom-Up position automaton and father automaton by considering a family of regular tree expressions. So, finding an uniform generator of regular tree expressions could be useful for a comparative study on average complexities.

After that, we introduced the extended tree expressions and we show how to compute a Brzozowski-like tree automata on these expressions (if it exists) using the Bottom-Up quotients. We managed to obtain a finite set of derivatives by applying the ACI rules of the sum. However, one can wonder: are they sufficient to prove that the constructed automaton is finite for all instances? Do we need another rules to affirm this finiteness? Moreover, we can wonder whether the choices of the tree_* functions during the computation of the derivative automaton impact the produced automaton.

Using the Bottom-Up interpretation in our constructions allows to solve the limit of the Top-Down ones.

Below, we recapitulate all the main notions realized in this thesis.

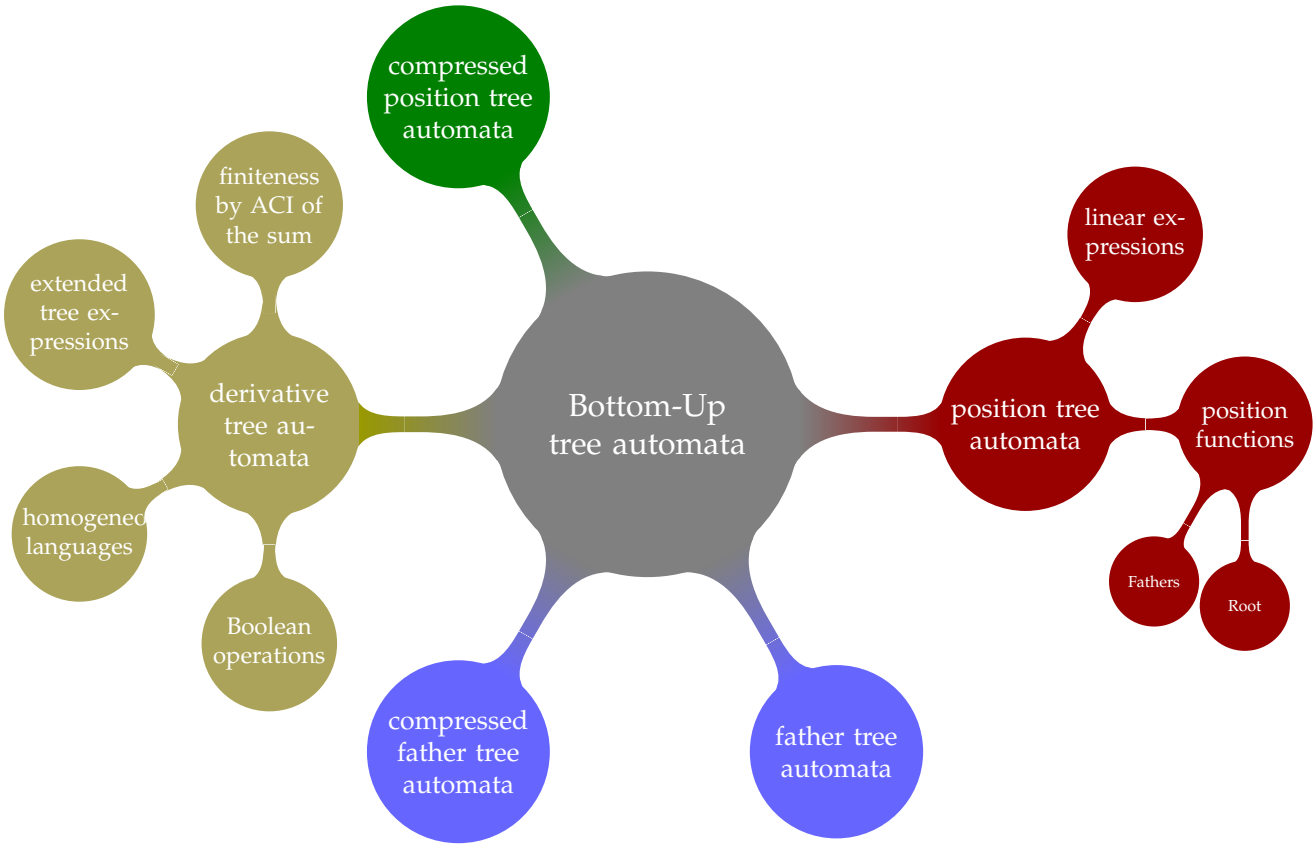


FIGURE 7.1: Recapitulation of the Bottom-Up constructions of tree automata.

There exist another class of tree automata much more generalized than the class studied in this thesis called: *hedge automata*. This latter recognizes unranked tree languages. So one can ask for the possibility to extend our study to the case of unranked tree automata. In other words: is it possible to translate regular unranked tree expressions to hedge automata recognizing the same unranked tree languages?

In the second part of our thesis, we proposed formulas in order to count the number of ranked trees over a finite sets of symbols $\Sigma = \{f^k, \alpha_1, \ldots, \alpha_s\}$ and $\Sigma = \{f_1^{k_1}, f_2^{k_2}, \alpha_1, \ldots, \alpha_s\}$. By respecting a given size, we found that this number represents the number of ways to place the symbols of rank ≥ 1 in the internal nodes of trees $\times$ the number of ways to place the α_i (for $1 \leq i \leq s$) in the leaves of trees at least once.

As a future work, we aim to generalize the formulas cited above and enumerate the ranked trees over $\Sigma = \{f_1^{k_1}, \ldots, f_r^{k_r}, \alpha_1, \ldots, \alpha_s\}$.

References

[1] J. Adámek and V. Trnková. *Automata and Algebras in Categories*. Mathematics and its Applications. Springer Netherlands, 1990.

[2] Valentin M. Antimirov. "Partial Derivatives of Regular Expressions and Finite Automaton Constructions". In: *Theor. Comput. Sci.* 155.2 (1996), pp. 291–319. DOI: 10.1016/0304-3975(95)00182-4.

[3] Dean N Arden. "Delayed-logic and finite-state machines". In: *2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961)*. IEEE. 1961, pp. 133–151.

[4] Samira Attou, Ludovic Mignot, and Djelloul Ziadi. "Extended tree expressions and their derivatives". In: *Eleventh Workshop on Non-Classical Models of Automata and Applications, NCMA 2019, Valencia, Spain, July 2-3, 2019*. Ed. by Rudolf Freund, Markus Holzer, and José M. Sempere. Österreichische Computer Gesellschaft, 2019, pp. 47–62.

[5] Samira Attou, Ludovic Mignot, and Djelloul Ziadi. "The bottom-up position tree automaton and its compact version". In: *International Conference on Implementation and Application of Automata*. Springer. 2018, pp. 59–70.

[6] Samira Attou, Ludovic Mignot, and Djelloul Ziadi. "The Bottom-Up Position Tree Automaton and the Father Automaton". In: *International Journal of Foundations of Computer Science* (2020), pp. 1–18.

[7] Jean-Christophe Aval. "Multivariate fuss–catalan numbers". In: *Discrete Mathematics* 308.20 (2008), pp. 4660–4669.

[8] Vitold Belevitch. "Summary of the history of circuit theory". In: *Proceedings of the IRE* 50.5 (1962), pp. 848–855.

[9] Gérard Berry and Ravi Sethi. "From Regular Expressions to Deterministic Automata". In: *Theor. Comput. Sci.* 48.3 (1986), pp. 117–126. DOI: 10.1016/0304-3975(86)90088-5.

[10] Béatrice Bouchou et al. "Schema Evolution for XML: A Consistency-Preserving Approach". In: *MFCS*. Vol. 3153. Lecture Notes in Computer Science. Springer, 2004, pp. 876–888.

[11] Anne Brüggemann-Klein. "Regular expressions into finite automata". In: *Theoretical Computer Science* 120.2 (1993), pp. 197–213.

[12] Anne Brüggemann-Klein and Derick Wood. "One-Unambiguous Regular Languages". In: *Inf. Comput.* 140.2 (1998), pp. 229–253.

[13] Janusz A. Brzozowski. "Derivatives of Regular Expressions". In: *J. ACM* 11.4 (1964), pp. 481–494. DOI: 10.1145/321239.321249.

[14] Janusz A Brzozowski and Edward J McCluskey. "Signal flow graph techniques for sequential circuit state diagrams". In: *IEEE Transactions on Electronic Computers* 2 (1963), pp. 67–76.

[15] Pascal Caron and Djelloul Ziadi. "Characterization of Glushkov automata". In: *Theor. Comput. Sci.* 233.1-2 (2000), pp. 75–90.

[16] A. Cayley. "A theorem on trees". In: *Quart. J. Math. 23 (1889)* 2 (1889), pp. 376–378.

[17] Jean-Marc Champarnaud and Djelloul Ziadi. "Canonical derivatives, partial derivatives and finite automaton constructions". In: *Theor. Comput. Sci.* 289.1 (2002), pp. 137–163. DOI: 10.1016/S0304-3975(01)00267-5.

[18] Jean-Marc Champarnaud and Djelloul Ziadi. "From C-Continuations to New Quadratic Algorithms for Automaton Synthesis". In: *IJAC* 11.6 (2001), pp. 707–736. DOI: 10.1142/S0218196701000772.

[19] Jean-Marc Champarnaud et al. "Bottom-Up Quotients for Tree Languages". In: *Journal of Automata, Languages and Combinatorics* 22.4 (2017), pp. 243–269.

[20] Jean-Marc Champarnaud et al. "Random Generation Models for NFAs". In: *J. Autom. Lang. Comb.* 9.2/3 (2004), pp. 203–216. DOI: 10.25596/jalc-2004-203. URL: https://doi.org/10.25596/jalc-2004-203.

[21] Noam Chomsky. "Three models for the description of language". In: *IRE Trans. Inf. Theory* 2.3 (1956), pp. 113–124. DOI: 10.1109/TIT.1956.1056813. URL: https://doi.org/10.1109/TIT.1956.1056813.

[22] Loek Gerard Willem Antoine Cleophas. "Tree algorithms: two taxonomies and a toolkit". In: (2008).

[23] H. Comon et al. *Tree Automata Techniques and Applications*. Available on: http://www.grappa.univ-lille3.fr/tata. release October, 12th 2007. 2007.

[24] SJ Dilworth and SR Mane. "Applications of Fuss-Catalan numbers to success runs of Bernoulli trials". In: *Journal of Probability and Statistics* 2016 (2016).

[25] Nikolaus I Fuss. "Solutio quaestionis, quot modis polygonum n laterum in polygona m laterum, per diagonales resolvi queat". In: *Nova Acta Academiae Sci. Petropolitanae* 9 (1791), pp. 243–251.

[26] John P. Gallagher, Mai Ajspur, and Bishoksan Kafle. "Optimised determinisation and completion of finite tree automata". In: *J. Log. Algebr. Meth. Program.* 95 (2018), pp. 1–16.

[27] John P. Gallagher, Kim S. Henriksen, and Gourinath Banda. "Techniques for Scaling Up Analyses Based on Pre-interpretations". In: *ICLP*. Vol. 3668. Lecture Notes in Computer Science. Springer, 2005, pp. 280–296.

[28] V. M. Glushkov. "The abstract theory of automata". In: *Russian Mathematical Surveys* 16 (1961), pp. 1–53.

[29] Younes Guellouma and Hadda Cherroun. "Efficient Implementation for Deterministic Finite Tree Automata Minimization". In: *J. Comput. Inf. Technol.* 24.4 (2016), pp. 311–322. URL: http://cit.fer.hr/index.php/CIT/article/view/2867.

[30] Younes Guellouma et al. "From Tree Automata to String Automata Minimization". In: *Theory Comput. Syst.* 62.5 (2018), pp. 1203–1222. DOI: 10.1007/s00224-017-9815-4. URL: https://doi.org/10.1007/s00224-017-9815-4.

[31] Silvia Heubach, Nelson Y. Li, and Toufik Mansour. "Staircase tilings and k-Catalan structures". In: *Discrete Mathematics* 308.24 (2008), pp. 5954–5964. DOI: 10.1016/j.disc.2007.11.012.

[32] Peter Hilton and Jean Pedersen. "Catalan numbers, their generalization, and their uses". In: *Math Intelligencer* 13.2 (1991), pp. 64–75.

[33] Johanna Högberg, Andreas Maletti, and Jonathan May. "Backward and forward bisimulation minimization of tree automata". In: *Theor. Comput. Sci.* 410.37 (2009), pp. 3539–3552. DOI: 10.1016/j.tcs.2009.03.022. URL: https://doi.org/10.1016/j.tcs.2009.03.022.

[34] David A Huffman. "The synthesis of sequential switching circuits". In: *Journal of the franklin Institute* 257.3 (1954), pp. 161–190.

[35] Lucian Ilie and Sheng Yu. "Follow automata". In: *Inf. Comput.* 186.1 (2003), pp. 140–162.

[36] Reza Kahkeshani. "A Generalization of the Catalan Numbers". In: *Journal of Integer Sequences* 16 (2013), Article 13.6.8.

[37] S. Kleene. "Representation of events in nerve nets and finite automata". In: *Automata Studies* Ann. Math. Studies 34 (1956). Princeton U. Press, pp. 3–41.

[38] Stephen Cole Kleene. *Representation of events in nerve nets and finite automata*. Tech. rep. RAND PROJECT AIR FORCE SANTA MONICA CA, 1951.

[39] Donald Ervin Knuth. *The art of computer programming, , Volume III, 2nd Edition*. Addison-Wesley, 1998. ISBN: 0201896850. URL: https://www.worldcat.org/oclc/312994415.

[40] Donald Ervin Knuth. *The art of computer programming, Volume I: Fundamental Algorithms, 3rd Edition*. Addison-Wesley, 1997. ISBN: 0201896834. URL: https://www.worldcat.org/oclc/312910844.

[41] Thomas Koshy. *Catalan Numbers with Applications*. Oxford: Oxford University Press, 2009.

[42] Dietrich Kuske and Ingmar Meinecke. "Construction of tree automata from regular expressions". In: *RAIRO - Theor. Inf. and Applic.* 45.3 (2011), pp. 347–370. DOI: 10.1051/ita/2011107.

[43] É. Laugerotte, N. Ouali Sebti, and D. Ziadi. "From Regular Tree Expression to Position Tree Automaton". In: *LATA 2013*. 2013, pp. 395–406. DOI: 10.1007/978-3-642-37064-9_35. URL: http://dx.doi.org/10.1007/978-3-642-37064-9_35.

[44] Warren S McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.

[45] R. F. McNaughton and H. Yamada. "Regular Expressions and State Graphs for Automata". In: *IEEE Transactions on Electronic Computers* 9 (Mar. 1960), pp. 39–57.

[46] Robert McNaughton and Hisao Yamada. "Regular expressions and state graphs for automata". In: *IRE transactions on Electronic Computers* 1 (1960), pp. 39–47.

[47] Ludovic Mignot, Nadia Ouali Sebti, and Djelloul Ziadi. "An Efficient Algorithm for the Equation Tree Automaton via the k-C-Continuations". In: *Language, Life, Limits - 10th Conference on Computability in Europe, CiE 2014, Budapest, Hungary, June 23-27, 2014. Proceedings*. 2014, pp. 303–313. DOI: 10.1007/978-3-319-08019-2_31.

[48] Ludovic Mignot, Nadia Ouali Sebti, and Djelloul Ziadi. "*K*-Position, Follow, Equation and *K*-C-Continuation Tree Automata Constructions". In: *Proceedings 14th International Conference on Automata and Formal Languages, AFL 2014, Szeged, Hungary, May 27-29, 2014*. 2014, pp. 327–341. DOI: 10.4204/EPTCS.151.23.

[49] Ludovic Mignot, Nadia Ouali Sebti, and Djelloul Ziadi. "Tree Automata Constructions from Regular Expressions: a Comparative Study". In: *Fundam. Inform.* 156.1 (2017), pp. 69–94. DOI: 10.3233/FI-2017-1598.

[50] Cyril Nicaud. "Etude du comportement en moyenne des automates finis et des langages rationnels". PhD thesis. Paris 7, 2000.

[51] Cyril Nicaud. "On the Average Size of Glushkov's Automata". In: *LATA*. Vol. 5457. Lecture Notes in Computer Science. Springer, 2009, pp. 626–637.

[52] Jean-Luc Ponty, Djelloul Ziadi, and Jean-Marc Champarnaud. "A New Quadratic Algorithm to Convert a Regular Expression into an Automaton". In: *Automata Implementation, First International Workshop on Implementing Automata, WIA '96, London, Ontario, Canada, August 29-31, 1996, Revised Papers*. Ed. by Darrell R. Raymond, Derick Wood, and Sheng Yu. Vol. 1260. Lecture Notes in Computer Science. Springer, 1996, pp. 109–119. DOI: 10.1007/3-540-63174-7_9. URL: https://doi.org/10.1007/3-540-63174-7_9.

[53] John Riordan. "The Enumeration of Trees by Height and Diameter". In: *IBM J. Res. Dev.* 4.5 (1960), pp. 473–478. DOI: 10.1147/rd.45.0473. URL: https://doi.org/10.1147/rd.45.0473.

[54] Marcel Paul Schützenberger. "On finite monoids having only trivial subgroups". In: *Inf. Control.* 8.2 (1965), pp. 190–194.

[55] MANOJ KUMAR SONI, AMIT SONI, and DEEPAK BANSAL. "SOME GEOMETRIC PROPERTIES OF ANALYTIC SERIES WHOSE COEFFICIENTS ARE RECIPROCAL OF FUSS-CATALAN NUMBERS". In: *Electronic Journal of Mathematical Analysis and Applications* 6.2 (2018), pp. 246–254.

[56] Richard P Stanley. *Catalan addendum*. 2008.

[57] Richard P Stanley. *Catalan numbers*. Cambridge University Press, 2015.

[58] Richard P. Stanley and Sergey P. Fomin. *Enumerative Combinatorics*. Vol. 2. Cambridge University Press, 1999.

[59] James A. Storer. *An Introduction to Data Structures and Algorithms*. Birkhäuser Basel, 2002. ISBN: 978-1-4612-0075-8.

[60] Robert Endre Tarjan. *Data structures and network algorithms*. SIAM, 1983.

[61] James W. Thatcher and Jesse B. Wright. "Generalized Finite Automata Theory with an Application to a Decision Problem of Second-Order Logic". In: *Mathematical Systems Theory* 2.1 (1968), pp. 57–81.

[62] Ken Thompson. "Programming techniques: Regular expression search algorithm". In: *Communications of the ACM* 11.6 (1968), pp. 419–422.

[63] Alan Mathison Turing. "On computable numbers, with an application to the Entscheidungsproblem". In: *Proceedings of the London mathematical society* 2.1 (1937), pp. 230–265.

[64] Bapiraju Vinnakota and V. V. Bapeswara Rao. "Enumeration of Binary Trees". In: *Inf. Process. Lett.* 51.3 (1994), pp. 125–127.

[65] John Von Neumann. "First Draft of a Report on the EDVAC, 30 June 1945". In: *Contract No W* (1945).