Des Codes Barres pour les Langages Rationnels

Ludovic Mignot

▶ **To cite this version:**

Ludovic Mignot. Des Codes Barres pour les Langages Rationnels. Théorie et langage formel [cs.FL]. Université de Rouen, 2010. Français. NNT : . tel-00794580

HAL Id: tel-00794580
https://theses.hal.science/tel-00794580

Submitted on 26 Feb 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université de Rouen U.F.R. des Sciences et Techniques

École Doctorale Sciences Physiques, Mathématiques et de l'Information pour l'Ingénieur

Thèse

présentée en vue d'obtenir le grade de
Docteur en « Informatique »

Des Codes Barres pour les Langages Rationnels

par
Ludovic Mignot

Soutenue Publiquement le : 15 Octobre 2010

Devant le jury composé de :

BASSINO Frédérique	PU	LIPN	Rapporteur
CARON Pascal	MCF	LITIS	Co-Directeur de thèse
CHAMPARNAUD Jean-Marc	PU	LITIS	Directeur de thèse
LOMBARDY Sylvain	PU	IGM	Rapporteur
NICAUD Cyril	MCF	IGM	Examinateur
PIN Jean-Éric	DR	CNRS LIAFA	Examinateur
ZIADI Djelloul	PU	LITIS	Examinateur

Laboratoire d'Informatique, de Traitement de l'Information et des Systèmes

Résumé

Les expressions rationnelles et les automates finis sont des objets mathématiques permettant de représenter les langages rationnels. Le lien entre ces structures est le sujet de nombreux thèmes de recherche. Chacun de ces modèles présentent avantages et inconvénients. Nous nous proposons d'établir de nouveaux opérateurs, les multi-tildes-barres, permettant de créer un modèle d'expression se situant entre la structure d'automate et celle d'expression rationnelle simple, utilisant l'union, la concaténation, et l'étoile de Kleene. Les multi-tildes-barres sont basées sur des opérations relativement simples sur les langages, l'ajout et l'élimination du mot vide. Nous étendons les méthodes de conversion classiques entre expressions rationnelles simples et automates finis aux expressions utilisant ces nouveaux opérateurs. Nous montrons également que le pouvoir de factorisation de ces nouvelles expressions est exponentiellement plus grand que celui des expressions rationnelles simples.

Mots-clés : Automates Finis, Langages Rationnels, Expressions Rationnelles, Méthodes de Conversion, Mot Vide, Concision des Expressions.

Abstract

Regular expressions and finite automata represent regular languages. The link between these two mathematical structures is the subject of many research topics. Each one of these models gets advantages and inconveniences. In this document, new regular operators are defined, the multi-tildes-bars, which create a new expression model falling in the range of finite automata to simple regular expressions, which uses concatenation, union, and Kleene's star. Multi-tildes-bars are based on simple regular operations, namely the adjunction or the elimination of the empty word. Several classical conversions between simple regular expressions and finite automata are extended to these new operators. Finally, we show that these new expressions have an exponential factorization power with respect to the one of simple regular expressions.

Keywords : Finite Automata, Regular Languages, Regular Expressions, Conversion Methods, Empty Word, Short Expressions.

Table des matières

Remerciements

La façon de remercier dépend de ce que l'on reçoit.
Pierre Dac

C'est d'après cette citation que je souhaite remercier de nombreuses personnes, qui m'ont apporté tant de choses durant ces trois années de doctorat.

Je remercie tout d'abord Frédérique Bassino et Sylvain Lombardy d'avoir accepté de rapporter mon manuscrit de thèse. Je remercie également Cyril Nicaud, Jean-Éric Pin et Djelloul Ziadi d'avoir accepté d'examiner ma soutenance de thèse.

Je souhaite remercier tous les membres du LITIS et plus particulièrement les membres de l'équipe Combinatoire et Algorithmes, ainsi que les membres du Département d'Informatique, pour leur accueil très chaleureux lors de mon arrivée dans le monde universitaire. Un grand merci à Christophe Hancart et Martine Léonard qui m'ont inculqué l'art de la pédagogie.

Le contenu de mes travaux de recherche prend sa source dans les travaux de deux grands chercheurs, Pascal Caron et Jean-Marc Champarnaud, qui m'ont fait l'honneur de diriger ma thèse.

C'est lors de mon stage de Master 2 que j'ai découvert la caractérisation des automates de Glushkov, due à Pascal Caron et Djelloul Ziadi. Pascal, dirigeant ce stage, m'a ainsi ouvert les portes du monde de la recherche, et plus particulièrement de *sa* recherche. Il s'en est suivi cette thèse, durant laquelle Pascal fut le lien entre l'ensemble de l'équipe et moi, thésard tout frais et tout naïf. C'est en grande partie grace à lui que mon intégration dans ce nouveau monde, dans lequel de nombreux doctorants se retrouvent totalement perdu, se fit sans le moindre problème. Pascal m'a ainsi beaucoup donné, et comme l'a dit Albert Einstein :

La valeur d'un homme tient dans sa capacité à donner
et non dans sa capacité à recevoir.

Un grand merci à Pascal pour son soutien et toute son aide sans lesquels je n'aurai pas pu réaliser cette thèse.

Jean-Marc Champarnaud est le plus grand chercheur que j'ai pu rencontrer. Son humilité extrême ainsi que son perfectionnisme furent pour moi la meilleure des formations à la recherche. Travailler avec lui fut un grand honneur pour moi. Il réussit durant ces années à me transmettre sa grande passion pour la recherche. Sa disponibilité fut un point important de la réussite de ces travaux. Toujours là au moindre problème, toujours présent pour répondre à mes questions plus tordues les unes que les autres. Son énorme culture scientifique et sa maîtrise du monde des automates m'a permis de ne pas m'égarer dans les voies sans issue sur lesquelles j'ai failli m'engager tant de fois durant ces années.

On ne devient grand qu'à apprendre d'un maître. (Yves Thériault)

Plus que des remerciements, c'est la fierté d'avoir eu cet excellent directeur de thèse que je souhaite lui adresser à travers ces quelques mots.

Un grand merci à toute ma famille, et plus particulièrement à mes parents, qui m'ont toujours soutenu dans mes choix et qui m'ont toujours démontrer leur soutien et leur confiance.

Je n'oublie pas John, Brahim, Thomas, Hadrien, Mikaël, Nico, Guillaume, et toutes nos soirées...

Et enfin, à Magalie, qui partage ma vie et qui a réussi à me supporter pendant ces trois années : c'est grace à toi si j'ai réussi ce doctorat.

Introduction

La science consiste à passer d'un étonnement à un autre.
Aristote

Seuls les mots sont aptes à rendre compte du rien.
Marc Gendron

L'être humain a toujours souhaité modéliser par les mathématiques le monde qui l'entoure. Dès l'Antiquité, les philosophes tentaient de comprendre le lien entre le monde du ressenti, le monde du perçu, et les constructions mathématiques. Tout au long de l'Histoire, les penseurs tentèrent de modéliser par le biais de machines, dispositifs théoriques ou non, la réalité qu'ils pouvaient alors percevoir.

En 1936, Turing proposa dans [45] une machine théorique particulièrement puissante, la machine de Turing, pouvant calculer ce que toute autre machine peut calculer. Toute machine de Turing est définie par un 6-tuplet $M = (\Gamma, 0, Q, i, F, \delta)$ avec :

(-) Γ un ensemble de lettres,
(-) $0 \in \Gamma$ le symbole blanc,
(-) Q un ensemble fini d'états,
(-) $i \in Q$ l'état initial,
(-) $F \subset Q$ les états finaux,
(-) $\delta \subset Q \times \Gamma \times Q \times \Gamma \times \{\leftarrow, \rightarrow\}$ la table d'actions.

Le fonctionnement de M est le suivant : une tête de lecture/écriture se positionne sur un ruban de longueur infini constitué de cases adjacentes contenant chacune un symbole de $\Gamma \setminus \{0\}$. La machine de Turing se trouve au départ dans l'état initial i. La table d'actions définit alors une suite d'actions en fonction de l'état de la machine et du symbole lu en entrée par la tête de lecture/écriture. Ainsi, toute action $(q, a, q', b, \leftarrow)$ (resp. $(q, a, q', b, \rightarrow)$) peut être exécutée lorsque la machine est dans l'état q et que le symbole lu par la tête de lecture est a. Son exécution place la machine dans l'état q', inscrit b à la place de a et déplace la tête de lecture/écriture d'une case vers la gauche (resp. vers la droite). La machine stoppe lorsqu'elle est dans un état terminal, et le résultat est alors inscrit sur le ruban. Turing démontra également l'existence d'une *super*-machine de Turing, la

machine de Turing universelle, capable de simuler le comportement de toute machine de Turing. Puisqu'une machine de Turing peut être représentée par une chaîne de caractères, on peut inscrire alors sur un ruban infini toutes les machines de Turing et former une machine de Turing universelle. L'article de Turing ([45]) est considéré comme l'article fondateur de la théorie des automates. En effet, cette branche des mathématiques fut développée pour comprendre et expliquer la capacité de calcul des machines de Turing.

Vingt ans plus tard, Kleene, dans [30], fut le premier a démontrer l'équivalence entre certaines machines de Turing particulières (les automates finis, machines de Turing sans ruban mémoire) et les grammaires régulières. Une grammaire est un objet mathématique permettant d'engendrer un ensemble de mots à partir de règles prédéfinies. La régularité d'une grammaire est une propriété définie sur ses règles. Ainsi, Kleene démontra que tout langage généré par une grammaire régulière peut être reconnu par une machine finie, et sans mémoire qui plus est. Par conséquent, différents types de mémoires furent étudiés sur des machines à nombre d'états fini afin de reconnaître d'autres types de langages : mémoire par une ou plusieurs piles, par registres, *etc...* Cependant, on s'aperçut que chaque type de machines reconnaissait un type de langages s'inscrivant dans une hiérarchie à quatre niveaux, où chaque langage d'un niveau k appartient également au niveau $k-1$. Cette hiérarchie avait été proposée par le linguiste Chomsky en 1956 dans [21] démontrant l'existence de quatre types de grammaires et de quatre uniquement. Il fut montré alors que pour chacun de ces types de grammaire, il existe un type d'automate reconnaissant le langage généré par une grammaire de ce type. Par exemple, les langages rationnels (de type 3) sont générés par des grammaires rationnelles (ou régulières) et reconnus par les automates à états finis. Ces langages sont potentiellement infinis, et leur description peut sembler particulièrement complexe. Afin de décrire plus facilement les langages rationnels furent créées les expressions rationnelles.

En 1943, McCulloch et Pitts modélisèrent dans [34] le système nerveux et neuronal par le biais d'automates simples. Kleene introduit alors dans [30] une représentation simple de ces objets, utilisant des opérations d'union, de concaténation et un nouvel opérateur révolutionnaire, l'étoile (portant désormais son nom, étoile de Kleene). En 1959, Rabin et Scott traitèrent mathématiquement toutes ces notions dans [41], article qui leur vaudra un prix Turing. Les expressions rationnelles furent intégrées dans divers domaines, en programmation par exemple, permettant de décrire très facilement des motifs particuliers générant des actions données (langage PERL ou parser LEX par exemple).

De nombreux auteurs se posèrent alors la question de la complexité, en espace et en temps, des algorithmes basés sur les structures d'automates et d'expressions rationnelles, mais aussi de la complexité de conversion

d'une des ces structures en l'autre.

Ainsi, sur les automates finis, la caractérisation de minimalité d'un automate fut réalisé sur un type d'automate particulier, les automates déterministes, par le biais d'une opération et d'une seule, le quotient d'un langage par un mot. Cependant, caractériser structurellement le plus petit automate (non-déterministe) reconnaissant un langage donné est toujours un sujet ouvert (voir [29]).

Sur les expressions rationnelles, aucune caractérisation des expressions rationnelles minimales n'a encore été proposée. Contrairement aux automates, il fut montré que pour passer d'une expression à une expression syntaxiquement différente dénotant le même langage, l'ensemble des règles de réécriture était potentiellement infini. En effet, Redko montra en 1964 dans [42] qu'il n'existe pas d'axiomatisation finie complète équationnelle. Salomaa proposa dans [44] la première axiomatisation finie complète, mais non-équationnelle. Kozen proposa en 1991 dans [31] (voir aussi [32]) une axiomatisation finie quasi-équationnelle.

Certains essayèrent alors de montrer les différentes bornes de la largeur d'une expression rationnelle en fonction du nombre d'états d'un automate reconnaissant le langage dénoté (voir [25]), les bornes atteintes s'avérant jusqu'à exponentielles.

La conversion d'une expression rationnelle en un automate à états finis fut une question rapidement résolue : indépendamment, Glushkov dans [24] et McNaughton et Yamada dans [35] proposèrent une méthode transformant une expression rationnelle possédant n lettres en un automate à $(n+1)$ états en temps $O(n^3)$. Brüggemann-Klein proposa dans [8] un prétraitement en temps $O(n)$ sur l'expression afin de construire l'automate en temps $O(n^2)$. Champarnaud, Ponty et Ziadi proposèrent dans [40] une construction en temps quadratique de l'automate de Glushkov.

La complexité spatiale de cette conversion fut également traitée et améliorée. Ainsi, différentes méthodes furent élaborées afin de calculer un automate plus petit, comme les méthodes des follows de Illie et Yu ([28]) ou par dérivation partielle d'Antimirov ([2]).

La conversion inverse, transformant un automate en une expression fut résolue par la construction inductive d'expressions. Cependant, un même algorithme appliqué à deux automates isomorphes ne produit pas les mêmes expressions, et la taille des expressions produites peut s'avérer exponentielle. Han et Wood proposèrent dans [26] de réaliser des coupes horizontales et/ou verticales sur les automates, produisant alors des automates dont la conversion en expression donne un résultat plus petit. Cette méthode fut améliorée par Ahn et Han dans [1]. Différents auteurs s'intéressèrent alors à comparer ces algorithmes en fixant des ordres particuliers ([43]).

Nous souhaitons ici réaliser une réduction du nombre de lettres d'une expression rationnelle par une approche différente, en créant de nouveaux

opérateurs basés sur des opérations rationnelles simples, à savoir l'ajout ou l'élimination du mot vide.

Nous montrerons que le pouvoir de factorisation engendré par les combinaisons d'ajouts et d'éliminations du mot vide est exponentiel. Nous réaliserons également une étude théorique des extensions à nos opérateurs des constructions classiques.

Le Chapitre 1 page 17 présente les notions de base utilisées tout au long de ce mémoire. Nous détaillons les définitions et les différentes propriétés des objets mathématiques que nous manipulerons tout au long de ce document.

Le Chapitre 2 page 39 est un état de l'art des différentes méthodes de conversion, que ce soit d'une expression rationnelle simple en un automate fini ou d'un automate fini en une expression rationnelle. Les méthodes de construction d'automate depuis une expression rationnelle simple sont toutes étendues aux nouveaux opérateurs dans le Chapitre 6. Les méthodes de conversion d'automate en expression se classent en deux familles : la première contient les méthodes de calcul d'une expression depuis un automate quelconque. Cette famille nous permet d'étudier les différentes notions manipulées lors de ces calculs. La seconde famille n'est constituée que d'une unique méthode, qui réalise la transformation inverse de la construction de Glushkov et qui répond à la question suivante : soit A un automate ; existe-t-il une expression rationnelle E dénotant $L(A)$ telle que l'automate des positions de E soit isomorphe à A ? Cette méthode est étendue aux nouveaux opérateurs dans le Chapitre 6.

Le Chapitre 3 page 69 introduit les opérateurs de multi-tildes et de multi-barres, nouveaux opérateurs basés sur des opérations rationnelles simples. Les multi-barres sont des opérateurs construits à partir de l'opération d'élimination du mot vide d'un langage rationnel. Leurs opérateurs duaux, les multi-tildes, sont quant à eux basés sur l'opération d'ajout du mot vide à un langage rationnel. Nous montrons dans ce chapitre que ces opérateurs possèdent des propriétés fondamentalement utiles, telles que la compatibilité avec la linéarisation et la rationalité.

Le Chapitre 4 page 87 présente une forme normalisée pour les multi-barres et pour les multi-tildes, permettant la réduction de la taille des opérateurs utilisés. Nous présentons également dans ce chapitre une extension de la construction de Glushkov à ces expressions. Cette conversion pour les multi-barres et les multi-tildes est la seule conversion étudiée pour ces expressions. La méthode de conversion inverse est quant à elle étudiée par le biais d'une famille plus large, les multi-tildes-barres, englobant les multi-barres et les multi-tildes.

Le Chapitre 5 page 107 présente les multi-tildes-barres, combinaisons simultanées des notions de multi-tildes et de multi-barres. Nous présentons également dans ce chapitre une forme particulière et plus descriptive, la forme totale, permettant de décrire l'action de l'opérateur sur tous les

sous-intervalles de l'intervalle sur lequel il est défini.

Dans le Chapitre 6 page 125 sont proposées les différentes extensions aux multi-tildes-barres des méthodes de conversion d'expression vers automate et d'automate vers expression.

Le Chapitre 7 page 155 démontre le pouvoir de factorisation des multi-tildes-barres par un exemple dans lequel le gain est exponentiel.

Enfin, le Chapitre 8 page 167 présente les différentes perspectives offertes par l'utilisation des multi-tildes-barres.

Chapitre 1

Préliminaires

La philosophie est toute entière préliminaire. À moins que ce ne soient les préliminaires qui soient déjà philosophie.
Vladimir Jankélévitch

Les idées audacieuses sont comme les pièces qu'on déplace sur un échiquier : on risque de les perdre mais elles peuvent aussi être l'amorce d'une stratégie gagnante.
Johann Wolfgang von Goethe

Sommaire

Dans ce chapitre, nous présentons les définitions, les propriétés et les algorithmes sur lesquels se base l'intégralité des notions définies dans les chapitres suivants. La Section 1.1 contient les différentes notations mathématiques utilisées tout au long de cette thèse. La Section 1.2 traite des notions de mot et de langage à la base de la théorie des automates. La Section 1.3 présente les notions d'automate et d'expression rationnelle, liées entre elles par le théorème de Kleene (Théorème 1.1, [30]). La Section 1.4 présente quelques propriétés, définitions et transformations sur les automates, telles que les transformations usuelles de déterminisation ou de standardisation. Différentes propriétés des expressions rationnelles sont données dans la Section 1.5. Les algorithmes de transformation définis dans ces sections sont présentés dans la Section 1.6.

1.1 Notations Mathématiques

Dans les chapitres suivants, nous utilisons des couples d'entiers représentant des intervalles. Ces intervalles modélisent différentes contraintes sur les structures que nous manipulons. Les notations suivantes permettent d'alléger l'écriture.

Soit i et f deux entiers positifs tels que $i \leq f$. Le couple (i, f) est dit *croissant*. L'ensemble des entiers $\{i, i+1, \ldots, f-1, f\}$ est noté $[\![i, f]\!]$. L'ensemble des couples d'entiers compris entre i et f est noté $[\![i, f]\!]^2$. L'ensemble des couples croissants d'entiers compris entre i et f, c'est-à-dire l'ensemble $\{(i', f') \in [\![i, f]\!]^2 \mid i' \leq f'\}$ est noté $[\![i, f]\!]^2_{\leq}$.

Soit S un ensemble. Le nombre des éléments de l'ensemble S, appelé *cardinal* de S, est noté $\#S$. Le seul ensemble de cardinal 0, l'*ensemble vide*, est représenté par $\emptyset$. Soit R une relation binaire sur S et soit $\cdot$ une opération interne dans S. La relation R est :

- *réflexive*	si $\forall x \in S$,	xRx.
- *symétrique*	si $\forall x, y \in S$,	$xRy \Rightarrow yRx$.
- *antisymétrique*	si $\forall x, y \in S$,	$(xRy \wedge yRx) \Rightarrow x = y$.
- *transitive*	si $\forall x, y, z \in S$,	$(xRy \wedge yRz) \Rightarrow xRz$.
- *stable à droite par rapport à* $\cdot$	si $\forall x, y, z \in S$,	$xRy \Rightarrow (x \cdot z)R(y \cdot z)$.

Si la relation R est réflexive, symétrique et transitive, R est une *relation d'équivalence*. Si R est une relation d'équivalence stable à droite par rapport à $\cdot$, R est une *congruence*. Si R est réflexive, antisymétrique et transitive, R est une *relation d'ordre*. Si tous les éléments de S sont comparables par R, c'est-à-dire si $\forall x, y \in S$, xRy ou yRx, R est un *ordre total* et S est *totalement ordonné* par R.

1.2 Mots et Langages

Dans la suite, nous manipulons des notions de théorie des langages. Cette branche des mathématiques utilise des objets particuliers que nous définissons dans cette section.

Un *alphabet* Σ est un ensemble de symboles tel que toute combinaison finie d'éléments de Σ se décompose d'une manière unique selon les éléments de Σ, c'est-à-dire que tout élément de Σ est insécable. Par la suite, nous n'utilisons que des alphabets finis.

Soit m un entier, et soit $w = (a_1, a_2 \ldots a_m)$ un élément de Σ^m. L'élément w est appelé *mot* sur l'alphabet Σ. Le mot w est considéré comme la concaténation des symboles le composant, à savoir $w = a_1 \cdot a_2 \cdots a_m$. S'il n'y a pas d'ambiguïté, le mot w est noté $w = a_1 a_2 \cdots a_m$ en omettant la concaténation. La *longueur* de w, notée $|w|$, est égale au nombre de ses lettres, ici m. L'unique mot de longueur 0, appelé *mot vide* est représenté par le symbole ε. L'ensemble des mots sur Σ est noté Σ^*.

Dans la suite, nous nous intéressons à des ensembles de mots, finis ou non, appelés *langages*. Ces langages ont été classifiés par Noam Chomsky en 1956 [21]. Nous utilisons ici une variété particulière de la classification de Chomsky, les langages de type 3, appelés également *langages rationnels*. La caractérisation de ces langages utilise les trois opérations suivantes :

Définition 1.1 *Soit L et L' deux langages sur un alphabet Σ. On définit les opérations* union, concaténation *et* étoile de Kleene *comme suit :*

$$\begin{aligned} \textit{union :} \quad & L \cup L' = \{w \mid w \in L' \ \vee \ w \in L'\}, \\ \textit{concaténation :} \quad & L \cdot L' = \{v = w \cdot w' \mid w \in L \ \wedge \ w' \in L'\}, \\ \textit{étoile de Kleene :} \quad & L^* = \textstyle\bigcup_{k \in \mathbb{N}} L^k \\ & \quad \textit{avec } L^0 = \{\varepsilon\} \\ & \quad \textit{et } L^k = L \cdot L^{k-1} \textit{ pour } k \geq 1. \end{aligned}$$

Définition 1.2 *Soit L un langage sur un alphabet Σ. Le langage L est dit* rationnel *s'il peut s'obtenir inductivement comme suit :*

$$L = \{a\}, \qquad L = \emptyset,$$
$$L = L_1 \cup L_2, \; L = L_1 \cdot L_2,$$
$$L = L_1^*,$$

avec L_1 et L_2 deux langages rationnels sur Σ et a un symbole de Σ.

L'ensemble des langages rationnels sur l'alphabet Σ est noté $\mathrm{Rat}(\Sigma^*)$.

L'ensemble $\mathrm{Rat}(\Sigma^*)$ est ainsi le plus petit ensemble contenant le langage vide $\emptyset$, les singletons $\{a\}$ avec $a \in \Sigma$, et qui est fermé par les trois opérations $\cup$, $\cdot$ et *.

Soit $L_1, \ldots, L_n$ des langages non vides sur un alphabet Σ. Considérons le langage $L = L_1 \cdots L_n$, w un mot de L et $w' = (w'_1, \cdots, w'_n)$ une séquence telle que pour tout k de $[\![1, n]\!]$, w'_k est dans L_k et $w'_1 \cdots w'_n = w$. La séquence w' est appelée *décomposition* de w sur $(L_1, \ldots, L_n)$. Cette décomposition n'est pas nécessairement unique (voir Exemple 1.1). Soit i et f deux entiers de $[\![1, n]\!]$ tels que $i \leq f$. Le mot $w'_i \cdot w'_{i+1} \cdots w'_f$ est un *ε-facteur* dans w' si et seulement si $w'_i \cdots w'_f = \varepsilon$. S'il existe $w'_{i-1} = \varepsilon$ (resp. s'il existe $w'_{f+1} = \varepsilon$), le mot $w'_i \cdots w'_f$ est *ε-prolongeable à gauche* (resp. *à droite*) dans w'. Si le mot $w'_i \cdots w'_f$ n'est pas ε-prolongeable (ni à gauche, ni à droite), alors il est *ε-maximal* dans w'. Afin d'alléger les notations, la notion de décomposition est traitée implicitement dans la suite : la décomposition w' est assimilée au mot w quand il n'y a aucune ambiguïté.

Exemple 1.1 *Considérons les langages rationnels L_1, L_2, L_3 et L définis par :*

$$L_1 = \{a\}^*, \; L_2 = \{\varepsilon, b\}, \; L_3 = \{a, b\}^* \textit{ et } L = L_1 \cdot L_2 \cdot L_3.$$

Soit $w = abb$ un mot de L. Plusieurs décompositions selon (L_1, L_2, L_3) sont possibles, telles que $w' = (a, b, b)$ ou $w'' = (\varepsilon, \varepsilon, abb)$. Remarquons qu'il n'y a aucun ε-facteur dans w', alors que les mots w''_1, w''_2 et $w''_1 \cdot w''_2$ sont des ε-facteurs dans w''. De plus, w''_1 et w''_2 sont ε-prolongeables dans w'', et $w''_1 \cdot w''_2$ est ε-maximal dans w''.

1.3 Automates et Expressions Rationnelles

Un langage rationnel est un ensemble potentiellement infini de mots. Plusieurs représentations de ces objets mathématiques existent. Dans la suite, nous utilisons les automates finis et les expressions rationnelles, définis dans cette section.

Un *automate* est un quintuplet $A = (\Sigma, Q, I, F, \delta)$ avec :
- Σ un alphabet, appelé *alphabet de l'automate*,
- Q l'ensemble des *états* de l'automate,
- $I \subset Q$ l'ensemble des *états initiaux* de l'automate,
- $F \subset Q$ l'ensemble des *états finaux* ou *terminaux* de l'automate,
- $\delta \subset Q \times \Sigma \times Q$ l'ensemble des *transitions* de l'automate.

Un *automate non-déterministe à nombre d'états fini* (*NFA*) est un automate sur un alphabet fini possédant un nombre fini d'états. Les états d'un automate peuvent être représentés graphiquement par des cercles reliés entre eux par des flèches étiquetées par des lettres. Certains de ces cercles possèdent des représentations particulières, puisque les états correspondants peuvent être initiaux ou terminaux. Par exemple, l'automate fini $A_1 = (\Sigma_1, Q_1, I_1, F_1, \delta_1)$ défini par
- $\Sigma_1 = \{a, b\}$,
- $Q_1 = \{1, 2, 3\}$,
- $I_1 = \{1\}$,
- $F_1 = \{3\}$,
- $\delta_1 = \{(1, a, 2),\ (2, a, 3), (1, b, 3),\ (3, b, 1)\}$,

admet la représentation graphique de la Figure 1.1. Tout état initial (comme ici l'état 1) est marqué par une flèche entrante, et tout état final (comme ici l'état 3) par un double cercle ou par une flèche sortante.

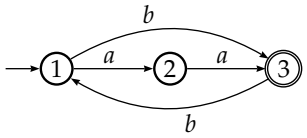


FIGURE 1.1 – *Représentation Graphique d'un Automate.*

Soit $A = (\Sigma, Q, I, F, \delta)$ un automate. L'ensemble de transitions δ est équivalent à une fonction de $Q \times \Sigma$ vers 2^Q définie par :

$$q' \in \delta(q, a) \Leftrightarrow (q, a, q') \in \delta.$$

Le caractère fonctionnel et le caractère ensembliste de δ sont utilisés dans la suite, et le passage de l'un à l'autre est particulièrement aisé.

Soit $Q' \subset Q$ un ensemble d'états, soit $a \in \Sigma$ une lettre de l'alphabet et soit w un mot de Σ^*. La fonction δ peut être étendue de $2^Q \times \Sigma^*$ vers 2^Q comme suit :

$$\delta(Q', \varepsilon) = Q',\ \ \delta(Q', a) = \bigcup_{q \in Q'} \delta(q, a),$$
$$\delta(Q', a \cdot w) = \delta(\delta(Q', a), w).$$

Le langage reconnu par l'automate, noté $L(A)$ est l'ensemble des mots qui sont l'étiquette d'un chemin partant d'un état initial et arrivant à un état final : $L(A) = \{w \in \Sigma^* \mid \delta(I, w) \cap F \neq \emptyset\}$. Par exemple, l'automate A_1 de la Figure 1.1 reconnaît le langage $L(A_1) = \{aa, b\} \cdot \{baa, bb\}^*$.

L'ensemble des langages sur un alphabet Σ reconnus par un NFA est noté $\mathrm{Rec}(\Sigma^*)$. Ces langages sont rationnels. Cette propriété est démontrée par l'utilisation des *expressions rationnelles simples*, reliées aux NFAs par le théorème de Kleene ([30]).

Définition 1.3 *Une* expression rationnelle simple *E sur un alphabet Σ est définie inductivement par :*

$$E = \emptyset, \quad E = \varepsilon, \quad E = a,$$
$$E = (F + G), \quad E = (F \cdot G), \quad E = (F^*).$$

avec $a \in \Sigma$, et F et G deux expressions rationnelles simples.

Par souci de clarté, les parenthèses peuvent être omises lorsqu'il n'y a pas d'ambiguïté. Ainsi, l'expression rationnelle simple $E = (a + (b^*))$ est notée $E = a + b^*$.

Une expression rationnelle simple décrit un ensemble de mots. Le langage $L(E)$ décrit par l'expression rationnelle simple E est appelé *langage dénoté* par E.

Définition 1.4 *Soit E une expression rationnelle simple sur un alphabet Σ. Le* langage dénoté *par E est défini inductivement par :*

$$L(\emptyset) = \emptyset, \quad L(\varepsilon) = \{\varepsilon\}, \quad L(a) = \{a\},$$
$$L(F + G) = L(F) \cup L(G), \quad L(F \cdot G) = L(F) \cdot L(G), \quad L(F^*) = L(F)^*,$$

avec $a \in \Sigma$, et F et G deux expressions rationnelles simples.

Tout langage dénoté par une expression rationnelle simple est rationnel. Et réciproquement, pour tout langage rationnel L donné, il existe une expression rationnelle simple dénotant L. Par exemple, le langage reconnu par le NFA A_1 de la Figure 1.1 est dénoté par l'expression rationnelle simple $E_1 = (a \cdot a + b) \cdot (b \cdot a \cdot a + b \cdot b)^*$. L'opérateur de concaténation peut être omis ; ainsi, l'expression E_1 peut s'écrire $E_1 = (aa + b)(baa + bb)^*$. Cette équivalence entre langage rationnel et langage dénoté par une expression rationnelle simple s'établit comme suit :

Lemme 1.1 *Soit L un langage sur un alphabet Σ. Le langage L est rationnel si et seulement s'il existe une expression rationnelle simple E telle que E dénote L.*

Démonstration. Par induction, posons :

$$\begin{array}{ll} L = \emptyset & E_L = \emptyset \\ L = \{a\} & E_L = a \\ L = L' \cup L'' & E_L = E_{L'} + E_{L''} \\ L = L' \cdot L'' & E_L = E_{L'} \cdot E_{L''} \\ L = (L')^* & E_L = (E_{L'})^* \end{array}$$

Avec L' et L'' deux langages rationnels,
$E_{L'}$ et $E_{L''}$ deux expressions rationnelles simples,
et a dans Σ.

Selon la Définition 1.2, la Définition 1.3 et la Définition 1.4, dans tous ces cas, L et $L(E_L)$ sont égaux. Ainsi, pour tout langage L, il existe une expression rationnelle simple E_L dénotant L et pour toute expression rationnelle simple E_L, le langage dénoté par E_L est rationnel. □

Par extension, une expression est dite *rationnelle* si elle n'utilise que des opérateurs rationnels ; rappelons qu'un opérateur est *rationnel* si et seulement si $\mathrm{Rat}(\Sigma^*)$ est fermé pour cet opérateur. Par exemple, puisque l'ensemble $\mathrm{Rat}(\Sigma^*)$ est fermé pour les opérateurs booléens (intersection, complémentaire, différence ensembliste), toute expression utilisant ces opérateurs est rationnelle.

Le théorème suivant énonce l'équivalence du pouvoir de description des expressions rationnelles et des automates finis.

Théorème 1.1 (Kleene, [30]) *Soit Σ un alphabet. Alors :*

$$\mathrm{Rec}(\Sigma^*) = \mathrm{Rat}(\Sigma^*).$$

Démonstration. Les méthodes constructives permettant de démontrer les deux sens de l'inclusion sont présentées dans le Chapitre 2.

$\mathrm{Rat}(\Sigma^*) \subset \mathrm{Rec}(\Sigma^*)$: voir Section 2.1.

$\mathrm{Rec}(\Sigma^*) \subset \mathrm{Rat}(\Sigma^*)$: voir Section 2.2. □

1.4 Propriétés des Automates

Dans cette section, différentes propriétés des automates sont présentées : propriétés élémentaires comme l'accessibilité ou la coaccessibilité, et propriétés fondamentales, comme le déterminisme ou la minimalité. Ces notions permettront de travailler dans les chapitres suivants avec des automates plus simples à traiter. Ces différentes propriétés mènent alors à des algorithmes de conversion, et nous montrons que ces conversions conservent les langages.

1.4.1 Langages Gauche et Droit d'un État d'un NFA

Considérons un NFA A reconnaissant un langage L. En plus du langage L, on peut considérer les langages reconnus par des automates obtenus en modifiant les états initiaux ou terminaux de A. En particulier, on peut calculer l'ensemble des mots qui sont l'étiquette d'un chemin partant d'un état initial et terminant sur un état q donné (*langage gauche de l'état q*), ou encore l'ensemble des mots qui sont l'étiquette d'un chemin terminant

sur un état terminal en partant d'un état q donné (*langage droit de l'état q*). Ces langages sont définis formellement comme suit :

Définition 1.5 *Soit $A = (\Sigma, Q, I, F, \delta)$ un automate et soit q un état de Q. Le* langage gauche *(resp.* langage droit*) de q dans A, noté $L_q^{\leftarrow}(A)$ (resp. $L_q^{\rightarrow}(A)$) est le langage défini par :*

$$L_q^{\leftarrow}(A) = \{w \in \Sigma^* \mid q \in \delta(I, w)\}$$
$$\textit{(resp. } L_q^{\rightarrow}(A) = \{w \in \Sigma^* \mid \delta(q, w) \cap F \neq \emptyset\}\textit{).}$$

Les langages gauche et droit des états d'un automate A décrivent le langage $L(A)$ d'une façon plus globale sur les états. Puisqu'un mot reconnu est l'étiquette d'un chemin allant d'un état initial à un état terminal (états non nécessairement distincts), ce chemin traverse nécessairement un état. Ainsi :

Lemme 1.2 *Soit $A = (\Sigma, Q, I, F, \delta)$ un automate. Alors :*

$$L(A) = \bigcup_{q \in Q} L_q^{\leftarrow}(A) \cdot L_q^{\rightarrow}(A).$$

Démonstration. Soit w un mot de $L(A)$. Alors w est l'étiquette d'un chemin réussi dans l'automate. Ainsi, $\delta(I, w) \cap F \neq \emptyset$, et donc il existe deux mots w' et w'' dans Σ^* tels que $w = w' \cdot w''$ et $\delta(\delta(I, w'), w'') \cap F \neq \emptyset$. Par conséquent il existe un état q de l'automate tel que $q \in \delta(I, w')$ et $\delta(q, w'') \cap F \neq \emptyset$. Le mot w appartient donc à l'ensemble $L_q^{\leftarrow}(A) \cdot L_q^{\rightarrow}(A)$. Finalement, le mot w appartient au langage $\bigcup_{q \in Q} L_q^{\leftarrow}(A) \cdot L_q^{\rightarrow}(A)$.

Soit $w \in \bigcup_{q \in Q} L_q^{\leftarrow}(A) \cdot L_q^{\rightarrow}(A)$. Ainsi il existe un état q' de l'automate tel que $w \in L_{q'}^{\leftarrow}(A) \cdot L_{q'}^{\rightarrow}(A)$. Par définition des langages gauche et droit, il existe deux mots, w' dans $L_{q'}^{\leftarrow}(A)$ et w'' dans $L_{q'}^{\rightarrow}(A)$ tels que $w = w' \cdot w''$ et $\delta(\delta(I, w'), w'') \cap F \neq \emptyset$. Finalement, $\delta(I, w) \cap F \neq \emptyset$ et $w \in L(A)$. □

Il n'est pas nécessaire de calculer les langages gauches et droits de tous les états d'un automate A pour calculer le langage reconnu par A. En effet, le langage de A se déduit des langages droits de ses états initiaux, ou des langages gauches de ses états terminaux, puisqu'un mot du langage est étiquette à la fois d'un chemin débutant sur un état initial et se terminant sur un état final. D'où le corollaire suivant :

Corollaire 1.1 *Soit A un automate, I son ensemble d'états initiaux et F son ensemble d'états finaux. Les deux propositions suivantes sont vérifiées :*

(1) $L(A) = \bigcup_{i \in I} L_i^{\rightarrow}(A)$,

(2) $L(A) = \bigcup_{f \in F} L_f^{\leftarrow}(A)$.

1.4.2 Propriétés Élémentaires des Automates

Soit $A = (\Sigma, Q, I, F, \delta)$ un NFA. Le NFA A est *accessible* (resp. *coaccessible*) si pour tout état q de Q, il existe un mot w de Σ^* tel que $q \in \delta(I, w)$

(resp. $\delta(q,w) \cap F \neq \emptyset$). Si A est à la fois accessible et coaccessible, l'automate A est *émondé*. Si pour toute lettre $a \in \Sigma$, pour tout état $q \in Q$, $\#\delta(q,a) \geq 1$, le NFA A est *complet*. Les notions d'accessibilité et de coaccessibilité se vérifient aisément sur les langages gauches et droits des états de l'automate. En fait, l'automate A est accessible (resp. coaccessible) si et seulement si pour tout état q de Q, le langage gauche (resp. droit) de q n'est pas vide. En se restreignant aux automates vérifiant chacune de ces propriétés, le pouvoir de description des automates en terme de langages rationnels reste inchangé.

Proposition 1.1 *Soit L un langage rationnel. Alors il existe un NFA A accessible (resp. coaccessible, émondé, complet) reconnaissant L.*

Démonstration. Voir Algorithme 1.1 et Lemme 1.10 pour l'accessibilité, Algorithme 1.2 et Lemme 1.11 pour la coaccessibilité, Algorithme 1.3 et Lemme 1.12 pour les automates émondés, Algorithme 1.4 et Lemme 1.13 pour les automates complets. □

1.4.3 Déterminisation d'un NFA

Dans un NFA, un même mot peut être l'étiquette de plusieurs chemins distincts débutant sur un état initial, ce qui implique que les langages gauches de deux états distincts d'un automate peuvent avoir une intersection non vide. Si cette propriété n'est pas vérifiée, le NFA est dit *déterministe*, et nous dirons que cet automate est un DFA. De plus, tout langage rationnel est reconnaissable par un DFA.

Définition 1.6 *Soit $A = (\Sigma, Q, I, F, \delta)$ un NFA. Le NFA A est un* automate déterministe *(*DFA*) si et seulement si les deux propriétés suivantes sont vérifiées :*

(1) *$\#I = 1$,*

(2) *Pour toute lettre $a \in \Sigma$, pour tout état $q \in Q$, $\#\delta(q,a) \leq 1$.*

Lemme 1.3 *Soit $A = (\Sigma, Q, \{i\}, F, \delta)$ un DFA. Pour tout couple d'états (q,q') de Q^2 tel que $q \neq q'$, on a :*

$$L_q^{\leftarrow}(A) \cap L_{q'}^{\leftarrow}(A) = \emptyset.$$

Démonstration. Supposons qu'il existe un mot w dans $L_q^{\leftarrow}(A) \cap L_{q'}^{\leftarrow}(A)$. Si $w = \varepsilon$, par définition des langages gauches, q et q' sont dans I. Contradiction avec l'unicité de l'état initial. Si w est dans Σ^+, puisque q et q' sont distincts, il existe nécessairement une décomposition $w = w' \cdot a \cdot w''$ avec w', w'' dans Σ^* et a dans Σ et trois états p, p', p'' de Q tel que $p \in \delta(i, w')$, $\{p', p''\} \subset \delta(p,a)$, $q \in \delta(p', w'')$ et $q' \in \delta(p'', w'')$. Contradiction avec $\#\delta(p,a) \leq 1$. □

Proposition 1.2 *Soit A un NFA. Alors il existe un DFA A' tel que $L(A) = L(A')$.*

Démonstration. Voir Algorithme 1.5 et Lemme 1.14. □

Exemple 1.2 *Soit A et A' les automates de la Figure 1.2 et de la Figure 1.3. On pourra vérifier que l'automate A' s'obtient en appliquant l'Algorithme 1.5 sur l'automate A.*

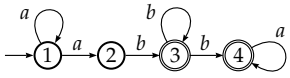


FIGURE 1.2 – *L'Automate A...*

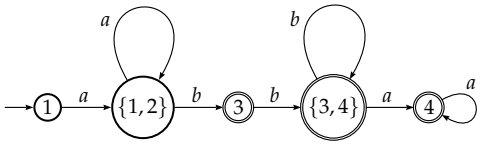


FIGURE 1.3 – *Et son Déterminisé A'.*

Le nombre d'états de l'automate produit par l'Algorithme 1.5 peut être exponentiel, comme l'illustre l'Exemple 1.3. En effet, un état du déterminisé est une partie de l'ensemble des états de l'automate de départ.

Exemple 1.3 *Considérons la famille des langages $L_n = \{a,b\}^* \cdot \{a\}\{a,b\}^n$ pour $n \geq 0$. Le langage L_n est reconnu par un NFA à $(n+2)$ états (cf. Figure 1.4). Le déterminisé de ce NFA produit par l'Algorithme 1.5 a 2^{n+1} états.*

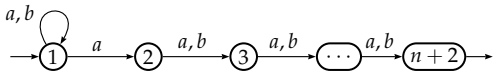


FIGURE 1.4 – *Le NFA à $(n+2)$ États Reconnaissant le Langage L_n.*

1.4.4 Minimisation d'un DFA

On peut se demander quelle est la taille (en nombre d'états) du plus petit DFA reconnaissant un langage rationnel donné. Le point le plus important de cette propriété est que l'automate minimal est unique. En effet, tous les automates déterministes minimaux reconnaissant un langage rationnel donné sont isomorphes. Une opération particulière est utilisée afin de montrer cette propriété, le quotient gauche d'un langage par une lettre. Remarquons que cette opération est récursivement calculable.

Définition 1.7 *Soit Σ un alphabet. Soit a une lettre de Σ et soit $L \subset \Sigma^*$ un langage. Le* quotient gauche *de L par a, noté $a^{-1}(L)$ est défini par :*

$$a^{-1}(L) = \{w' \in \Sigma^* \mid a \cdot w' \in L\}.$$

Lemme 1.4 *Soit L_1 et L_2 deux langages de $\mathrm{Rat}(\Sigma^*)$ et soit a et b deux lettres de Σ. L'opération a^{-1} se calcule récursivement comme suit :*

$$a^{-1}(\emptyset) = \emptyset, \qquad a^{-1}(L_1 \cup L_2) = a^{-1}(L_1) \cup a^{-1}(L_2),$$

$$a^{-1}(\{\varepsilon\}) = \emptyset, \qquad a^{-1}(L_1 \cdot L_2) = \begin{cases} a^{-1}(L_1) \cdot L_2 \cup a^{-1}(L_2) & \text{si } \varepsilon \in L_1, \\ a^{-1}(L_1) \cdot L_2 & \text{sinon,} \end{cases}$$

$$a^{-1}(\{b\}) = \begin{cases} \varepsilon \text{ si } a = b, \\ \emptyset \text{ sinon,} \end{cases} \qquad a^{-1}(L_1^*) = a^{-1}(L_1)L_1^*.$$

Démonstration. Par définition, $a^{-1}(L) = \{w' \in \Sigma^* \mid a \cdot w' \in L\}$. Pour les langages $\emptyset$, $\{\varepsilon\}$ et $\{b\} \subset \Sigma$, l'égalité est trivialement démontrée.

(a) Si $L = L_1 \cup L_2$,

$$\begin{aligned} a^{-1}(L_1 \cup L_2) &= \{w' \in \Sigma^* \mid a \cdot w' \in L_1 \cup L_2\} \\ &= \{w' \in \Sigma^* \mid a \cdot w' \in L_1 \ \vee \ a \cdot w' \in L_2\} \\ &= \{w' \in \Sigma^* \mid a \cdot w' \in L_1\} \cup \{w' \in \Sigma^* \mid a \cdot w' \in L_2\} \\ &= a^{-1}(L_1) \cup a^{-1}(L_2). \end{aligned}$$

(b) Si $L = L_1 \cdot L_2$,

$$\begin{aligned} a^{-1}(L_1 \cdot L_2) &= \{w' \in \Sigma^* \mid a \cdot w' \in L_1 \cdot L_2\} \\ &= \begin{cases} \{w' \in \Sigma^* \mid a \cdot w' \in L_1 \cdot L_2\} \\ \quad \cup \{w' \in \Sigma^* \mid a \cdot w' \in L_2\} & \text{si } \varepsilon \in L_1, \\ \{w' \in \Sigma^* \mid a \cdot w' \in L_1 \cdot L_2\} & \text{sinon,} \end{cases} \\ &= \begin{cases} \{w' = u \cdot v \in \Sigma^* \mid a \cdot u \in L_1 \ \wedge \ v \in L_2\} \\ \qquad \cup \{w' \in \Sigma^* \mid a \cdot w' \in L_2\} & \text{si } \varepsilon \in L_1, \\ \{w' = u \cdot v \in \Sigma^* \mid a \cdot u \in L_1 \ \wedge \ v \in L_2\} & \text{sinon,} \end{cases} \\ &= \begin{cases} \{u \in \Sigma^* \mid a \cdot u \in L_1\} \cdot L_2 \\ \qquad \cup a^{-1}(L_2) & \text{si } \varepsilon \in L_1, \\ \{u \in \Sigma^* \mid a \cdot u \in L_1\} \cdot L_2 & \text{sinon,} \end{cases} \\ &= \begin{cases} a^{-1}(L_1) \cdot L_2 \cup a^{-1}(L_2) & \text{si } \varepsilon \in L_1, \\ a^{-1}(L_1) \cdot L_2 & \text{sinon.} \end{cases} \end{aligned}$$

(c) Si $L = L_1^* = L_1^+ \cup \{\varepsilon\} = L_1 \cdot L_1^* \cup \{\varepsilon\}$,

$$\begin{aligned} a^{-1}(L_1^+ \cup \{\varepsilon\}) &= a^{-1}(L_1^+) \\ &= \{w' \in \Sigma^* \mid a \cdot w' \in L_1^+\} \\ &= \{w' = u \cdot v \in \Sigma^* \mid a \cdot u \in L_1 \ \wedge \ v \in L_1^*\} \\ &= \{u \in \Sigma^* \mid a \cdot u \in L_1\} \cdot L_1^* \\ &= a^{-1}(L_1) \cdot L_1^*. \end{aligned}$$

□

Cette opération de quotient par une lettre s'étend aisément à la notion de quotient par un mot. Pour un langage L sur un alphabet Σ et un mot w de Σ^*, on a :

$$w^{-1}(L) = \{w' \in \Sigma^* \mid w \cdot w' \in L\}.$$

Les formules de calcul récursif du quotient d'un langage rationnel par une lettre s'étendent au quotient par un mot de la façon suivante. Soit L un langage sur l'alphabet Σ et soit w un mot de Σ^*. Alors :

$$w^{-1}(L) = \begin{cases} L & \text{si } w = \varepsilon \\ w'^{-1}(a^{-1}(L)) & \text{si } w = a \cdot w' \text{ avec } a \in \Sigma \text{ et } w' \in \Sigma^* \end{cases}$$

À partir de ces définitions, on définit alors la relation suivante dans Σ^*, due à Myhill et Nerode ([38]).

Définition 1.8 *Soit L un langage (non nécessairement rationnel) sur un alphabet Σ et soit w_1 et w_2 deux mots de Σ^*. On note $\sim_L$ la relation :*

$$w_1 \sim_L w_2 \Leftrightarrow w_1^{-1}(L) = w_2^{-1}(L).$$

Cette relation est en fait une congruence, puisque pour tout w_1, w_2, w_3 de Σ^*, pour tout a de Σ :

- $\sim_L$ est réflexive : $w_1^{-1}(L) = w_1^{-1}(L)$,
- $\sim_L$ est symétrique : $w_1^{-1}(L) = w_2^{-1}(L) \Leftrightarrow w_2^{-1}(L) = w_1^{-1}(L)$,
- $\sim_L$ est transitive :

$$\left.\begin{array}{r} w_1^{-1}(L) = w_2^{-1}(L) \\ \text{et } w_2^{-1}(L) = w_3^{-1}(L) \end{array}\right\} \Rightarrow w_1^{-1}(L) = w_3^{-1}(L),$$

– $\sim_L$ est stable à droite par la concaténation :

$w_1 \sim_L w_2 \Rightarrow (w_1 \cdot a)^{-1}(L) = a^{-1}(w_1^{-1}(L)) = a^{-1}(w_2^{-1}(L)) = (w_2 \cdot a)^{-1}(L)$.

L'ensemble des mots de Σ^* équivalents à w selon la congruence $\sim_L$ est représenté par $[w]_{\sim_L}$. Les classes d'équivalence engendrées par cette congruence permettent de calculer l'*automate quotient*.

Définition 1.9 *Soit L un langage sur un alphabet Σ, et $A = (\Sigma, Q, I, F, \delta)$ l'automate défini par :*

- $Q = \{[w]_{\sim_L} \mid w \in \Sigma^*\}$,
- $I = \{[\varepsilon]_{\sim_L}\}$,
- $F = \{[w]_{\sim_L} \mid w \in L\}$,
- $\delta = \{([w]_{\sim_L}, a, [w']_{\sim_L}) \in Q \times \Sigma \times Q \mid [w']_{\sim_L} = [w \cdot a]_{\sim_L}\}$.

L'automate A est appelé automate quotient de L.

Par définition, $\delta([w]_{\sim_L}, a) = [w \cdot a]_{\sim_L}$, et par conséquent l'automate quotient est déterministe, accessible et complet. Montrons alors que l'automate quotient d'un langage L reconnaît le langage L et que le fait qu'il soit fini est une propriété des langages rationnels.

Lemme 1.5 *L'automate quotient A d'un langage L reconnait L.*

Démonstration. Posons $A = (\Sigma, Q, i, F, \delta)$. Par construction de A, pour w un mot de Σ^*, pour tout mot w' de $[w]_{\sim_L}$, alors $\delta([\varepsilon]_{\sim_L}, w') = [w]_{\sim_L}$ et ainsi $L^{\leftarrow}_{[w]_{\sim_L}}(A) = [w]_{\sim_L}$.

Soit w un mot de L. Alors il existe un état q de F tel que $q = [w]_{\sim_L}$. Par construction, w est dans $L^{\leftarrow}_q(A)$. Ainsi, w est dans $\bigcup_{q \in F} L^{\leftarrow}_q(A)$. Et selon le Corollaire 1.1, w appartient à $L(A)$.

Soit w un mot de $L(A)$. Alors il existe un chemin réussi d'étiquette w dans A. Ainsi il existe un état $q = [w']_{\sim_L}$ de F tel que $\delta(i, w) = q$. Par définition, si q est dans F, tout mot de $[w']_{\sim_L}$ est dans L, et par conséquent w est dans L. □

Théorème 1.2 (Myhill et Nerode ([38])) *Soit L un langage et soit A son automate quotient. Alors : A est un automate fini $\Leftrightarrow$ L est rationnel.*

Démonstration. Si A est un automate fini, selon le Théorème de Kleene, alors $L(A) = L$ est rationnel.

Si L est rationnel, soit A' un DFA complet reconnaissant L. Selon la définition de la relation $\sim_L$, pour tout état q de A', pour tout w, w' de $L^{\leftarrow}_q(A')$, $w \sim_L w'$. Puisque l'automate est complet, tout mot de Σ^* apparaît dans le langage gauche d'un état. L'union des langages gauches des états de A' est une partition de Σ^* plus fine que $\sim_L$. Puisqu'elle est d'indice finie, $\sim_L$ l'est aussi. □

L'automate quotient d'un langage rationnel est le DFA minimal en

nombre d'états reconnaissant L. En effet, on peut définir un morphisme surjectif de tout DFA reconnaissant L vers l'automate quotient de L. Par conséquent, deux DFA minimaux reconnaissant L sont isomorphes.

Lemme 1.6 *Soit L un langage et soit $A = (\Sigma, Q, I, F, \delta)$ son automate quotient. Considérons un automate $A' = (\Sigma, Q', I', F', \delta')$ déterministe complet et accessible tel que $L(A') = L$. Alors il existe une application surjective de Q' vers Q.*

Démonstration. Soit q' un état de Q'. Puisque A' est accessible, le langage $L^{\leftarrow}_{q'}(A')$ n'est pas vide. Selon la définition de la relation $\sim_L$, pour tous mots w, w' de $L^{\leftarrow}_{q'}(A')$, $w \sim_L w'$. Soit w un mot de $L^{\leftarrow}_{q'}(A')$. Considérons l'application f de Q' dans Q telle que $f(q') = [w]$. Puisque A' est déterministe et complet, tout mot de Σ^* est l'étiquette d'un chemin unique (non nécessairement réussi) de A'. Ainsi, pour tout état $[w]$ de Q, il existe un état q' de Q' tel que $q' = \delta'(i', w)$ et ainsi $f(q') = [w]$. Par conséquent, l'application f est surjective. □

Corollaire 1.2 *Soit L un langage rationnel et soit A son automate quotient. Alors A est le DFA minimal en nombre d'états reconnaissant L.*

Pour tout langage rationnel L, il existe un DFA canonique reconnaissant ce langage, l'automate quotient. Mais la construction de cet automate quotient est algorithmiquement complexe. Cependant, puisque pour tout DFA reconnaissant un langage L, il existe un morphisme surjectif vers l'automate quotient, on peut déduire de cette propriété une fusion d'états de l'automate par la congruence suivante :

Définition 1.10 *Soit A un DFA sur l'alphabet Σ et soit p et q deux états de cet automate. Les états p et q sont équivalents au sens de Myhill-Nerode si et seulement si :*

$$p \sim_M q \Leftrightarrow L^{\rightarrow}_p(A) = L^{\rightarrow}_q(A).$$

Lemme 1.7 *La relation $\sim_M$ est une congruence droite dans Q.*

Démonstration. Soit p, q, r des états d'un DFA A sur un alphabet Σ et soit w un mot de Σ^*.

- $\sim_{\mathbf{M}}$ est une relation réflexive : $L^{\rightarrow}_p(A) = L^{\rightarrow}_p(A)$.
- $\sim_{\mathbf{M}}$ est une relation symétrique : $L^{\rightarrow}_p(A) = L^{\rightarrow}_q(A) \Leftrightarrow L^{\rightarrow}_q(A) = L^{\rightarrow}_p(A)$.
- $\sim_{\mathbf{M}}$ est une relation transitive :
$$\left.\begin{array}{r} L^{\rightarrow}_p(A) = L^{\rightarrow}_q(A) \\ \text{et } L^{\rightarrow}_q(A) = L^{\rightarrow}_r(A) \end{array}\right\} \Rightarrow L^{\rightarrow}_q(A) = L^{\rightarrow}_r(A),$$
- $\sim_{\mathbf{M}}$ est une relation stable par l'action de l'alphabet sur les états :
$$p \sim_M q \Rightarrow L^{\rightarrow}_{\delta(p,a)}(A) = a^{-1}(L^{\rightarrow}_p(A)) = a^{-1}(L^{\rightarrow}_q(A)) = L^{\rightarrow}_{\delta(q,a)}(A)$$
$$\Rightarrow \delta(p,a) \sim_M \delta(q,a).$$

Ainsi, la relation $\sim_M$ est une congruence. □

On calcule alors à partir d'un DFA $A = (\Sigma, Q, \{i\}, F, \delta)$ un automate $A' = (\Sigma, Q', \{i'\}, F', \delta')$ quotienté par la congruence $\sim_M$. Cet automate est

défini par :
- $Q' = \{Q/_{\sim_M}\}$,
- $I' = \{[i]_{\sim_M}\}$,
- $F' = \{[p]_{\sim_M} \mid p \in F\}$,
- $\delta' = \{([p]_{\sim_M}, a, [q]_{\sim_M}) \in Q' \times \Sigma \times Q' \mid [q]_{\sim_M} = [\delta(p,a)]_{\sim_M}\}$.

Cet automate obtenu par fusion d'états du DFA de départ est isomorphe à l'automate quotient du langage. Et les classes d'équivalence de $\sim_M$ sont calculables pour un langage rationnel L à partir d'un DFA reconnaissant ce langage.

Lemme 1.8 *Soit L un langage rationnel, soit A son automate quotient et soit A' un DFA complet reconnaissant L. Alors A et $A'/_{\sim_M}$ sont isomorphes.*

Démonstration. $p \sim_M q \Leftrightarrow L_p^{\rightarrow}(A') = L_q^{\rightarrow}(A') \Leftrightarrow$ pour w dans $L_p^{\leftarrow}(A')$, pour w' dans $L_q^{\leftarrow}(A')$, $w^{-1}(L) = w'^{-1}(L) \Leftrightarrow$ pour w dans $L_p^{\leftarrow}(A')$, pour tout w' dans $L_q^{\leftarrow}(A')$ $w \sim_L w'$. Ainsi il existe une bijection ϕ définie par $\phi([p]_{\sim_M}) = [w]_{\sim_L}$ avec $w \in L_p^{\leftarrow}(A')$ et $\phi^{-1}([w]_{\sim_L}) = [\delta(i,w)]_{\sim_M}$. Par conséquent, les automates sont isomorphes. □

Proposition 1.3 *Soit L un langage rationnel et soit A un DFA reconnaissant L. Le DFA minimal de L est calculable à partir de A.*

Démonstration. Plusieurs algorithmes calculent les classes d'équivalence permettant d'obtenir le DFA minimal par fusion, tels que l'algorithme de Hopcroft ([27]) ou de Blum ([5]) de complexité $O(n \times \log(n))$, ou l'algorithme de Moore ([37]) de complexité $O(n^2)$ avec n le nombre d'états. □

Ainsi, un DFA est minimal si et seulement si les langages droits de ses états sont deux à deux distincts. Cette caractérisation a donné lieu à une construction exceptionnellement simple du DFA minimal à partir d'un NFA quelconque. Cette construction est due à Brzozowski ([9]), et utilise l'*automate retourné* :

Définition 1.11 *Soit $A = (\Sigma, Q, I, F, \delta)$ un NFA et soit $A' = (\Sigma, Q, I', F', \delta')$ le NFA défini par :*
- $I' = F$,
- $F' = I$,
- $\delta' = \{(q, a, q') \in Q \times \Sigma \times Q \mid (q', a, q) \in \delta\}$.

L'automate A' est appelé retourné *de A.*

Soit A un NFA, A' son retourné et q un état de A. Considérons un mot $w = a_1 \cdots a_m$ et son retourné $w' = a_m \cdots a_1$. Si w appartient au langage gauche de q dans A, alors w' appartient au langage droit de q dans A'. De même, si w appartient au langage droit de q dans A, alors w' appartient au langage gauche de q dans A'. En considérant les fonctions Det calculant le déterminisé d'un automate et Ret calculant le retourné d'un automate, on peut calculer le DFA minimal d'un langage comme suit :

Théorème 1.3 (Brzozowski ([9])) *Soit A un NFA. Soit $A' = \mathrm{Det}(\mathrm{Ret}(\mathrm{Det}(\mathrm{Ret}(A))))$. Alors A' est le DFA minimal de $L(A)$.*

Démonstration. Soit $A'' = \mathrm{Det}(\mathrm{Ret}(A))$. Puisque l'automate A'' est déterministe, les langages gauches de ses états sont deux à deux disjoints. En retournant l'automate A'', les langages droits de l'automate ainsi calculé sont alors deux à deux disjoints. Par déterminisation, les langages droits des états de A' sont alors des combinaisons différentes de langages disjoints. Ainsi, les langages droits de A' sont deux à deux distincts. Et puisque A' est déterministe, A' est le DFA minimal de $L(A)$. □

1.4.5 Standardisation et Homogénéisation

Dans les chapitres suivants, nous présentons des algorithmes permettant la conversion d'un automate en une expression, et réciproquement d'une expression vers un automate. Un de ces algorithmes est appliqué sur des automates standards et homogènes. Ces automates possèdent le même pouvoir de description que les NFA généraux.

Définition 1.12 *Soit $A = (\Sigma, Q, I, F, \delta)$ un NFA. Le NFA A est* standard *si et seulement s'il possède un unique état initial i non réentrant, c'est-à-dire qu'aucune transition n'a pour destination l'état initial i.*

Proposition 1.4 *Soit A un automate à n états. Alors il existe un automate standard A' avec au plus $(n+1)$ états tel que $L(A) = L(A')$.*

Démonstration. Voir Algorithme 1.6 et Lemme 1.15. □

Exemple 1.4 *Considérons l'automate A de la Figure 1.5 et l'automate A' de la Figure 1.6. L'automate A' est le standardisé de l'automate A. Puisque l'état 2 n'est plus accessible dans A', il est supprimé. De plus, l'état initial 0 de A' est terminal puisque l'état 3 l'est dans A.*

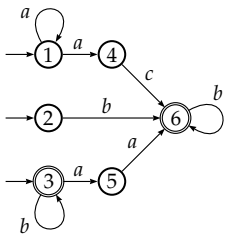


FIGURE 1.5 – *L'Automate A...*

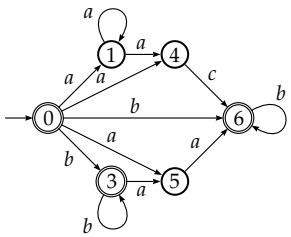


FIGURE 1.6 – *Et son Standardisé A'.*

Définition 1.13 *Soit $A = (\Sigma, Q, I, F, \delta)$ un NFA et soit $q \in Q$ un état de A. L'état q est homogène si et seulement si toutes les transitions dont la destination est q sont étiquetées par la même lettre.*

Définition 1.14 *Soit $A = (\Sigma, Q, I, F, \delta)$ un automate. L'automate A est homogène si et seulement si chacun de ses états est homogène.*

Proposition 1.5 *Soit A un automate à n états. Alors il existe un automate homogène A' avec au plus $(\#\Sigma \times n)$ états tel que $L(A) = L(A')$.*

Démonstration. Voir Algorithme 1.7 et Lemme 1.16. □

Exemple 1.5 *Considérons l'automate A de la Figure 1.7 et l'automate A' de la Figure 1.8. L'automate A' est l'homogénéisé de l'automate A et on pourra vérifier que les langages $L(A')$ et $L(A)$ sont égaux.*

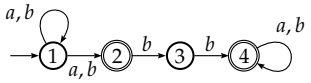


FIGURE 1.7 – *L'Automate A...*

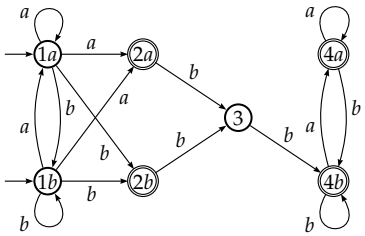


FIGURE 1.8 – *Et son Homogénéisé A'.*

1.5 Propriétés des Expressions Rationnelles

Dans la suite de cette thèse, nous nous intéressons à la réduction du nombre de lettres d'une expression rationnelle. Ce nombre est appelé *largeur alphabétique* de l'expression. La largeur alphabétique d'une expression E est notée $|E|$. Plusieurs expressions syntaxiquement différentes peuvent dénoter le même langage. Si E et E' sont deux expressions telles que $L(E) = L(E')$, les expressions E et E' sont *équivalentes* et on note $E \sim E'$. Si E et E' sont syntaxiquement équivalentes, on note $E = E'$. Si pour une expression E, il n'existe pas d'expression équivalente moins large, E est une *expression minimale*. Cette notion peut être restreinte à un ensemble d'opérateurs. Certaines règles permettent de transformer des expressions en expressions équivalentes. Dans ce but, tout au long de cette thèse seront utilisées différentes simplifications syntaxiques, appelées *quotients*. Le premier ensemble de simplifications défini utilise les égalités suivantes, en considérant E une expression rationnelle :

$$L(E + \emptyset) = L(\emptyset + E) = L(E), \quad L(E \cdot \emptyset) = L(\emptyset \cdot E) = \emptyset,$$
$$L(E \cdot \varepsilon) = L(\varepsilon \cdot E) = L(E).$$

Définition 1.15 *Soit E une expression. Alors :*

$$E + \emptyset \sim \emptyset + E \sim E, \quad E \cdot \emptyset \sim \emptyset \cdot E \sim \emptyset,$$
$$E \cdot \varepsilon \sim \varepsilon \cdot E \sim E.$$

Une expression E est *linéaire* si chacun de ses symboles apparaît une et une seule fois, et donc si $\#\Sigma_E = |E|$. D'où la notion *d'expression linéarisée* :

Définition 1.16 *Soit E une expression rationnelle sur un alphabet Σ. L'*expression linéarisée *de E, notée $E^{\#}$, est obtenue en remplaçant chacune des lettres de l'expression par sa position. Pour revenir à l'expression de départ est utilisé le morphisme h_E de $\Sigma_{E^{\#}}^*$ dans Σ^* tel que $\mathrm{h}_E(k)$ est la lettre en position k dans E. Toute expression linéarisée est linéaire puisque deux lettres ne peuvent pas avoir la même position.*

Exemple 1.6 *Soit $E = (a+b)\cdot a^* + a\cdot b^*$ une expression rationnelle simple. L'expression linéarisée de E est l'expression $E^{\#} = (1+2)\cdot 3^* + 4\cdot 5^*$.*

Considérons l'expression E de l'Exemple 1.6. Les langages $\mathrm{h}_E(L(E^{\#}))$ et $L(E)$ sont égaux. Cependant, cette propriété n'est pas nécessairement vérifiée par toute expression rationnelle et par sa linéarisée. En fait, cela dépend des opérateurs utilisés. Soit $E = a \cap a$ une expression rationnelle. Son expression linéarisée est l'expression $E^{\#} = 1 \cap 2$. Alors $L(E) = \{a\}$ et $L(E^{\#}) = \emptyset$, et ainsi, $\mathrm{h}_E(L(E^{\#})) \neq L(E)$. Cette remarque permet de définir une nouvelle notion, la compatibilité avec la linéarisation :

Définition 1.17 *Soit $\mathcal{O}$ un ensemble d'opérateurs. Les opérateurs de $\mathcal{O}$ sont* compatibles avec la linéarisation *si et seulement si pour toute expression E utilisant ces opérateurs, en notant $E^{\#}$ sa linéarisée, on a :*

$$L(E) = \mathrm{h}_E(L(E^{\#})).$$

L'ensemble d'opérateurs $\{+, \cdot, ^*\}$ utilisé par toute expression rationnelle simple est compatible avec la linéarisation :

Lemme 1.9 *Soit E une expression rationnelle simple. Alors :*

$$L(E) = \mathrm{h}_E(L(E^{\#})).$$

Démonstration. Par induction sur la structure, la largeur et le nombre d'opérateurs des expressions rationnelles simples. Lors de la linéarisation d'une expression, les opérateurs et par conséquent la structure de l'expression sont inchangés. Nous notons l'expression linéarisée $E^{\#}$ de $E = F + G$ (resp. $E = F\cdot G$, $E = F^*$) comme $E^{\#} = F' + G'$ (resp. $E^{\#} = F'\cdot G'$, $E^{\#} = (F')^*$). Puisque $E^{\#}$ est linéaire, les alphabets de F' et de G' sont disjoints.

$$\begin{aligned}
E = \varepsilon : \mathrm{h}_E(L(E^{\#})) &= \{\varepsilon\} \\
&= L(E)
\end{aligned}
\qquad
\begin{aligned}
E = F + G : \mathrm{h}_E(L(E^{\#})) &= \mathrm{h}_E(L(F' + G')) \\
&= \mathrm{h}_E(L(F')) \cup \mathrm{h}_E(L(G')) \\
&= L(F) \cup L(G) \\
&= L(E)
\end{aligned}$$

$$\begin{aligned}
E = \emptyset : \mathrm{h}_E(L(E^{\#})) &= \emptyset \\
&= L(E)
\end{aligned}
\qquad
\begin{aligned}
E = F\cdot G : \mathrm{h}_E(L(E^{\#})) &= \mathrm{h}_E(L(F'\cdot G')) \\
&= \mathrm{h}_E(L(F'))\cdot \mathrm{h}_E(L(G')) \\
&= L(F)\cdot L(G) \\
&= L(E)
\end{aligned}$$

$$\begin{aligned}
E = a : \mathrm{h}_E(L(E^{\#})) &= \mathrm{h}_E(L(a_k)) \\
&= \mathrm{h}_E(\{a_k\}) \\
&= \{a\} \\
&= L(E)
\end{aligned}
\qquad
\begin{aligned}
E = F^* : \mathrm{h}_E(L(E^{\#})) &= \mathrm{h}_E(L(F')^*) \\
&= \mathrm{h}_E(L(F'))^* \\
&= L(F)^* \\
&= L(E)
\end{aligned}$$

□

1.6 Algorithmes de Transformation

Algorithme 1.1 RendreAccessible

Entrées: $A = (\Sigma, Q, I, F, \delta)$ un NFA.
Sorties: $A' = (\Sigma', Q', I', F', \delta')$ un NFA accessible tel que $L(A) = L(A')$.

1: **File d'États** $L = \text{FileVide}()$; **Ensemble d'États** $M = \emptyset$
2: **État** q, x ; **Lettre** a
3: **si** $I = \emptyset$ **alors**
4: **État** $0 = \text{NouvelEtat}(0)$ {avec $0 \notin Q$} ; $I \leftarrow \{0\}$; $M \leftarrow \{0\}$
5: **sinon**
6: **pour tout** $q \in I$ **faire**
7: $\text{Enfiler}(q, L)$; $M \leftarrow M \cup \{q\}$
8: **fin pour**
9: **tantque** $\text{Non}(\text{EstFileVide}(\text{L}))$ **faire**
10: $x = \text{Defiler}(L)$
11: **pour tout** $a \in \Sigma$ **faire**
12: **pour tout** $q \in \delta(x, a)$ tel que $q \notin M$ **faire**
13: $M \leftarrow M \cup \{q\}$; $\text{Enfiler}(q, L)$
14: **fin pour**
15: **fin pour**
16: **fin tantque**
17: **finsi**
18: **return** $A' = (\Sigma, M, I, M \cap F, \delta \cap (M \times \Sigma \times M))$.

Lemme 1.10 *L'Algorithme 1.1 calcule à partir d'un NFA A un NFA A' accessible tel que $L(A') = L(A)$.*

Démonstration. Par construction, tout état q de Q appartient à M si et seulement s'il est accessible depuis un état initial. Ainsi, pour tout état q de $Q \setminus M$, le langage gauche de q est vide. L'élimination de cet état ne modifie pas le langage reconnu par l'automate, selon le Lemme 1.2. Finalement, tout état de Q' est accessible et $L(A') = L(A)$. □

Algorithme 1.2 RendreCoaccessible

Entrées: $A = (\Sigma, Q, I, F, \delta)$ un NFA.
Sorties: $A' = (\Sigma', Q', I', F', \delta')$ un NFA accessible tel que $L(A) = L(A')$.

1: **return** $A' = \text{Retourne}(\text{RendreAccessible}(\text{Retourne}(A)))$.

Lemme 1.11 *L'Algorithme 1.2 calcule à partir d'un NFA A un NFA A' coaccessible tel que $L(A') = L(A)$.*

Démonstration. Rendre Accessible le retourné d'un NFA, puis le retourner conserve le langage. De plus, puisque le retourné est accessible, les langages gauches de chacun des états sont différents de l'ensemble vide. Ainsi, dans le retourné A', les langages droits sont différents de l'ensemble vide. Finalement, le NFA A' est alors coaccessible. □

Algorithme 1.3 Emonder

Entrées: $A = (\Sigma, Q, I, F, \delta)$ un NFA.
Sorties: $A' = (\Sigma', Q', I', F', \delta')$ un NFA émondé tel que $L(A) = L(A')$.
1: $A' \leftarrow$ RendreAccessible(A) ; $A' \leftarrow$ RendreCoaccessible(A')
2: **return** A'.

Lemme 1.12 *L'Algorithme 1.3 calcule à partir d'un NFA A un NFA A' émondé tel que $L(A') = L(A)$.*

Démonstration. Selon l'Algorithme 1.1 et l'Algorithme 1.2, $L(A) = L(A')$ et pour chacun des états q de Q', les langages gauche et droit de q sont différents de l'ensemble vide. Par conséquent, l'automate A' est émondé. □

Algorithme 1.4 Compléter

Entrées: $A = (\Sigma, Q, I, F, \delta)$ un NFA.
Sorties: $A' = (\Sigma', Q', I', F', \delta')$ un NFA complet tel que $L(A) = L(A')$.
1: **État** $0 =$ NouvelEtat(0)
2: **Ensemble d'États** $Q' \leftarrow Q \cup \{0\}$ {avec $0 \notin Q$}
3: **Ensemble d'États** $I' \leftarrow I$; $F' \leftarrow F$
4: **Ensemble de Transitions** $\delta' \leftarrow \delta$
5: **État** q
6: **Lettre** a
7: **pour tout** $q \in Q'$ **faire**
8: **pour tout** $a \in \Sigma$ **faire**
9: **si** $\delta'(q, a) = \emptyset$ **alors**
10: $\delta'(q, a) \leftarrow \{0\}$
11: **finsi**
12: **fin pour**
13: **fin pour**
14: **return** $A' = (\Sigma, Q', I', F', \delta')$.

Lemme 1.13 *L'Algorithme 1.4 calcule à partir d'un NFA A un NFA A' complet tel que $L(A') = L(A)$.*

Démonstration. Un unique état 0 est ajouté, qui n'est pas coaccessible. Ainsi, le langage droit de 0 est vide. L'ajout des transitions vers cet état ne

modifie donc pas le langage. Ainsi, l'automate A' est complet par construction et $L(A') = L(A)$. □

Algorithme 1.5 Construire l'Automate des Parties

Entrées: $A = (\Sigma, Q, I, F, \delta)$ un NFA.
Sorties: $A' = (\Sigma', Q', I', F', \delta')$ un DFA tel que $L(A') = L(A)$.

Liste d'Ensemble d'États $L = \text{ListeVide}()$
Ensemble d'Ensemble d'États $Q' \leftarrow \{I\}$; $I' \leftarrow \{I\}$; $F' \leftarrow \emptyset$
Ensemble de Transitions $\delta' \leftarrow \emptyset$
Ensemble d'États X, Y
État z
Lettre a
Enfiler(I, L)
tantque NON(EstListeVide(L)) **faire**
  $X \leftarrow$ Defiler(L)
  pour tout $a \in \Sigma$ **faire**
    $Y \leftarrow \bigcup_{z \in X} \delta(z, a)$; $\delta'(X, a) \leftarrow Y$
    si $Y \notin Q'$ **alors**
      $Q' \leftarrow Q' \cup \{Y\}$; Enfiler(Y, L)
    finsi
    si $X \cap F \neq \emptyset$ **alors**
      $F' \leftarrow F' \cup \{X\}$
    finsi
  fin pour
fin tantque
return $A' = (\Sigma, Q', I', F', \delta')$.

Lemme 1.14 *L'Algorithme 1.5 calcule à partir d'un NFA A un DFA A' tel que $L(A') = L(A)$.*

Démonstration. Posons $A = (\Sigma, Q, I, F, \delta)$ et $A' = (\Sigma', Q', I', F', \delta')$. Par construction de A', $\#I' = 1$ et pour toute lettre $a \in \Sigma'$, pour tout état $q' \in Q'$, $\#\delta(q', a) \leq 1$. L'automate A' est donc déterministe.

Soit w un mot de $L(A)$. Si $w = \varepsilon$, il existe un i de I tel que $i \in F$. Par construction, l'état initial de A', l'état I, est aussi terminal. Ainsi, le mot w est dans $L(A')$. Si $w = a_1 \cdots a_n$, alors $\delta(I, w) \cap F \neq \emptyset$ et ainsi, il existe une suite d'états $(q_0, \ldots, q_n)$ de Q telle que $q_0 \in I$, $q_n \in F$ et pour tout entier $k \in [\![0, n-1]\!]$, $q_{k+1} \in \delta(q_k, a_{k+1})$. Soit i' l'état initial de A'. Par construction de A', $q_0 \in i'$. Posons la suite d'états $(q'_0, \ldots, q'_n)$ avec $q'_0 = i'$ et pour tout entier $k \in [\![1, n-1]\!]$, $q'_{k+1} = \delta(q'_k, a_{k+1})$. Par construction de A', puisque pour tout $k \in [\![0, n-1]\!]$, $q_{k+1} \in \delta(q_k, a_{k+1})$, on a alors pour tout $k \in [\![0, n]\!]$, $q_k \in q'_k$. Ainsi, $q'_k \cap F' \neq \emptyset$, et w est donc l'étiquette d'un chemin réussi dans A'. Par définition du langage reconnu par un automate, $w \in L(A')$.

Soit w un mot de $L(A')$. Puisque A' est déterministe, on a $I' = i'$ et $\#\delta(i', w) = 1$. Posons $q'_n = \delta(i', w)$. Par construction de A', q'_n est égal à l'ensemble $\{q \in Q \mid q \in \delta(I, w)\}$. Puisque $q'_n \in F'$, il existe un état $q \in q'_n$ tel que $q \in F$. Ainsi, $q \in \delta(I, w)$ et $q \in F$, et donc $w \in L(A)$.

Ainsi, l'automate A' est déterministe et $L(A') = L(A)$. □

Algorithme 1.6 Standardiser

Entrées: $A = (\Sigma, Q, I, F, \delta)$ un NFA.
Sorties: $A' = (\Sigma', Q', I', F', \delta')$ un NFA standard tel que $L(A') = L(A)$.

État $0 = \text{NouvelEtat}(0)$
Ensemble d'États $Q' \leftarrow Q \cup \{0\}$ {avec $0 \notin Q$}
Ensemble d'États $I' \leftarrow \{0\}$; $F' \leftarrow F$
Ensemble de transition $\delta' \leftarrow \delta$
État i
Lettre a
si $F \cap I \neq \emptyset$ **alors**
    $F' \leftarrow F' \cup \{0\}$
finsi
pour tout $i \in I$ **faire**
    pour tout $a \in \Sigma$ **faire**
        $\delta'(0, a) \leftarrow \delta'(0, a) \cup \delta(i, a)$
    fin pour
fin pour
return $A' = (\Sigma, Q', I', F', \delta')$

Lemme 1.15 *L'Algorithme 1.6 de standardisation calcule à partir d'un NFA A à n états un NFA standard A' avec au plus $(n+1)$ états tel que $L(A') = L(A)$.*

Démonstration. Posons $A' = (\Sigma, Q', I', F', \delta')$ et $A = (\Sigma, Q, I, F, \delta)$. Par construction, l'automate A' est standard et possède $n + 1$ états.

Soit w un mot de $L(A)$. Alors il existe un état initial i de I tel que $w \in L_i^{\rightarrow}$. Si $w = \varepsilon$, $i \in F$ et par construction, $0 \in F'$, et $w \in L(A')$. Si $w = a \cdot w'$, puisque $w \in L(A)$, il existe q dans Q tel que $q \in \delta(i, a)$ et $w' \in L_q^{\rightarrow}$. Par construction, q appartient à Q', $q \in \delta(0, a)$ et w' est dans le langage gauche de q. Ainsi, w est dans $L(A')$.

Soit w un mot de $L(A')$. Si $w = \varepsilon$, alors $0 \in F$ et par construction il existe un $i \in I$ tel que $i \in F$. Ainsi, $w \in L(A)$. Si $w = aw'$, alors il existe q dans Q' tel que $q \in \delta(0, a)$ et w' est dans le langage gauche de q. Par construction, q appartient à Q, et il existe un $i \in I$ tel que $q \in \delta(i, a)$. Puisque w' est dans le langage gauche de q, $w \in L(A)$. □

Algorithme 1.7 Homogénéiser

Entrées: $A = (\Sigma, Q, I, F, \delta)$ un NFA.
Sorties: $A' = (\Sigma', Q', I', F', \delta')$ un NFA homogène tel que $L(A') = L(A)$.

Ensemble d'états $Q' \leftarrow Q$; $I' \leftarrow I$; $F' \leftarrow F$
Fonction de transition $\delta' \leftarrow \delta$
États q', q'_a
Lettre a
pour tout $q' \in Q'$ tel que q' n'est pas homogène **faire**
  pour tout $a \in \Sigma$ **faire**
    $q'_a = \text{CreerNouvelEtat}(q', a)$; $Q' \leftarrow Q' \cup q'_a$
    si $q' \in F'$ **alors**
      $F' \leftarrow F' \cup \{q'_a\}$
    finsi
    si $q' \in I'$ **alors**
      $I' \leftarrow I' \cup \{q'_a\}$
    finsi
    pour tout $b \in \Sigma$ **faire**
      $\delta'(q'_a, b) \leftarrow \delta'(q', b)$
    fin pour
  fin pour
fin pour
pour tout $q' \in Q'$ tel que q' n'est pas homogène **faire**
  $I' \leftarrow I' \setminus \{q'\}$; $F' \leftarrow F' \setminus \{q'\}$
  pour tout $(p, a, p') \in \delta'$ tel que $p = q'$ ou $p' = q'$ **faire**
    $\delta' \leftarrow \delta' \setminus (p, a, p')$
  fin pour
  $Q' \leftarrow Q' \setminus \{q'\}$
fin pour
return $A' = (\Sigma, Q', I', F', \delta')$

Lemme 1.16 *L'Algorithme 1.7 d'homogénéisation calcule à partir d'un NFA A un NFA homogène A' tel que $L(A') = L(A)$.*

Démonstration. Posons $A = (\Sigma, Q, I, F, \delta)$ et $A' = (\Sigma, Q', I', F', \delta')$. Par construction, tout état de A qui n'est pas homogène est éclaté en au plus $(\#\Sigma)$ états homogènes. Ainsi, $\#Q' \leq n \times \#\Sigma$.

Soit w un mot de $L(A)$. Si $w = \varepsilon$, il existe i dans I tel que $i \in F$. Si i est homogène, $i \in I'$, $i \in F'$ et donc $w \in L(A')$. Si i n'est pas homogène, alors pour tout i_a avec $a \in \Sigma$, $i_a \in I'$ et $i_a \in F'$ et ainsi, $w \in L(A')$. Si w se décompose en $a_1 \cdots a_m$, alors il existe une suite d'états de Q $(q_0, \ldots, q_m)$ telle que $q_0 \in I$, $q_m \in F$ et telle que pour tout $k \in [\![1, m]\!]$, q_k est dans $\delta(q_{k-1}, a_k)$. Par construction, il existe une suite d'états $(q'_0, \ldots, q'_m)$ telle que pour tout $k \in [\![0, m]\!]$, $q'_k = q_k$ si q_k est homogène, $q'_k = (q_k)_{a_k}$ sinon. Et

par construction, cette suite est telle que $q'_0 \in I'$, $q'_m \in F'$ et telle que pour tout $k \in [\![1, m]\!]$, $q'_k \in \delta(q'_{k-1}, a_k)$. Ainsi, $w \in L(A')$.

Soit w un mot de $L(A')$. Si $w = \varepsilon$, il existe i dans I' tel que $i \in F'$. Par construction, soit l'état i provient de l'éclatement d'un état initial et terminal de A, soit l'état i est un état initial et terminal de l'automate A. Dans les deux cas, $w \in L(A)$. Si $w = a_1 \cdots a_m$, alors il existe une suite d'états de Q' $(q'_0, \ldots, q'_m)$ telle que $q'_0 \in I'$, $q'_m \in F'$ et telle que pour tout $k \in [\![1, m]\!]$, $q'_k \in \delta(q'_{k-1}, a_k)$. Par construction, chacun des états de cette suite soit provient de l'éclatement d'un état non homogène, soit est également dans Q. Ainsi, il existe une suite d'états $(q_0, \ldots, q_m)$ de Q telle que pour tout $k \in [\![0, m]\!]$, $q'_k = q_k$ si q_k est homogène, $q'_k = (q_k)_{a_k}$ sinon. Et par construction, $q_0 \in I$, $q_m \in F$ et pour tout $k \in [\![1, m]\!]$, $q_k \in \delta('_{k-1}, a_k)$. Ainsi, $w \in L(A)$. □

Chapitre 2

Conversions

Si l'on sait exactement ce qu'on va faire, à quoi bon le faire ?
Pablo Picasso

Rien ne se perd, rien ne se crée, tout se transforme.
Antoine-Laurent de Lavoisier

Sommaire

Dans ce chapitre, nous présentons un état de l'art des méthodes de conversion d'une expression rationnelle E dénotant un langage L en un NFA reconnaissant L (Section 2.1), et réciproquement d'un NFA reconnaissant un langage L en une expression rationnelle dénotant L (Section 2.2).

2.1 Convertir une Expression Rationnelle en un NFA

Les NFA et les expressions rationnelles possèdent le même pouvoir d'expressivité dans leur représentation des langages rationnels. En effet, il est possible de calculer à partir d'une expression rationnelle un automate reconnaissant le langage dénoté par l'expression. De même, depuis tout automate il est possible de calculer une expression rationnelle dénotant le langage reconnu par cet automate. Ce second sens sera présenté dans la section suivante. Cinq méthodes différentes permettant de calculer un NFA depuis une expression rationnelle sont présentées dans cette section.

2.1.1 L'Automate des Positions

La première méthode étudiée est la méthode de Glushkov [24] (méthode définie indépendamment par McNaughton et Yamada dans [35]), permettant de calculer depuis une expression à n lettres un automate à $(n+1)$ états. Cette méthode utilise des calculs sur l'expression linéarisée. L'expression est analysée en vue d'obtenir cinq ensembles (les ensembles Pos, Null, First, Last et Follow) permettant de lier la structure d'expression rationnelle à celle d'un automate. Chacun de ces ensembles permet d'expliciter certaines propriétés des symboles de l'expression en considérant leurs positions, d'où l'utilité de l'expression linéarisée.

Considérons une expression rationnelle E. La fonction $\mathrm{Pos}(E)$ explicite les différentes positions de l'expression linéarisée de E. La fonction $\mathrm{Null}(E)$ retourne le singleton $\{\varepsilon\}$ si le mot vide est contenu dans le langage dénoté par E, l'ensemble vide $\emptyset$ sinon. La fonction $\mathrm{First}(E)$ retourne les positions des lettres pouvant débuter un mot du langage. La fonction $\mathrm{Last}(E)$ retourne les positions des lettres pouvant terminer un mot du langage. Enfin, la fonction $\mathrm{Follow}(E,x)$, où x est un élément de $\mathrm{Pos}(E)$, retourne les positions pouvant succéder à la lettre en position x dans un mot du langage dénoté par l'expression E.

Soit E une expression rationnelle, $E^{\#}$ son expression linéarisée et x un élément de $\Sigma_{E^{\#}}$. Les cinq ensembles sont définis formellement de la façon suivante :

$$
\begin{aligned}
\mathrm{Pos}(E) &= \Sigma_{E^{\#}} \\
\mathrm{Null}(E) &= \begin{cases} \{\varepsilon\} & \text{si } \varepsilon \in L(E), \\ \emptyset & \text{sinon.} \end{cases} \\
\mathrm{First}(E) &= \{x \in \mathrm{Pos}(E) \mid \exists w' \in \Sigma^*_{E^{\#}}, x \cdot w' \in L(E^{\#})\} \\
\mathrm{Last}(E) &= \{x \in \mathrm{Pos}(E) \mid \exists w' \in \Sigma^*_{E^{\#}}, w' \cdot x \in L(E^{\#})\} \\
\mathrm{Follow}(E,x) &= \{x' \in \mathrm{Pos}(E) \mid \exists w', w'' \in \Sigma^*_{E^{\#}}, w' \cdot x \cdot x' \cdot w'' \in L(E^{\#})\}
\end{aligned}
$$

TABLE 2.1 – *Définitions des Fonctions de Glushkov.*

Exemple 2.1 *Soit $E = (a+b)^*a + b^*$ une expression rationnelle et soit $E^{\#} = (1+2)^*3 + 4^*$ son expression linéarisée. Les fonctions de Glushkov pour l'expression E sont les suivantes :*

$$
\begin{array}{ll}
\mathrm{Pos}(E) = \{1,2,3,4\} & \mathrm{Follow}(E,1) = \{1,2,3\} \\
\mathrm{Null}(E) = \{\varepsilon\} & \mathrm{Follow}(E,2) = \{1,2,3\} \\
\mathrm{First}(E) = \{1,2,3,4\} & \mathrm{Follow}(E,3) = \emptyset \\
\mathrm{Last}(E) = \{3,4\} & \mathrm{Follow}(E,4) = \{4\}
\end{array}
$$

On pourra vérifier que les mots $\mathrm{h}_E(\varepsilon) = \varepsilon$, $\mathrm{h}_E(11213) = aabaa$, $\mathrm{h}_E(3) = a$, $\mathrm{h}_E(22123) = bbaba$ et $\mathrm{h}_E(44) = bb$ appartiennent au langage $L(E)$ selon les formules de la Table 2.1.

Les expressions rationnelles sont définies de façon inductive. Cette propriété permet de calculer récursivement ces cinq fonctions. Chacune de ces fonctions peut se définir sur chacun des opérateurs et atomes de la struc-

ture d'une expression rationnelle. Le calcul récursif sur une expression linéaire s'effectue comme suit :

$$\begin{array}{rlrl} \mathrm{Pos}(\emptyset) &= \emptyset & \mathrm{Pos}(F+G) &= \mathrm{Pos}(F) \cup \mathrm{Pos}(G) \\ \mathrm{Pos}(\varepsilon) &= \emptyset & \mathrm{Pos}(F \cdot G) &= \mathrm{Pos}(F) \cup \mathrm{Pos}(G) \\ \mathrm{Pos}(a) &= \{a\} & \mathrm{Pos}(F^*) &= \mathrm{Pos}(F) \end{array}$$

TABLE 2.2 – *Calcul de la Fonction* Pos.

$$\begin{array}{rlrl} \mathrm{Null}(\emptyset) &= \emptyset & \mathrm{Null}(F+G) &= \mathrm{Null}(F) \cup \mathrm{Null}(G) \\ \mathrm{Null}(\varepsilon) &= \{\varepsilon\} & \mathrm{Null}(F \cdot G) &= \mathrm{Null}(F) \cap \mathrm{Null}(G) \\ \mathrm{Null}(a) &= \emptyset & \mathrm{Null}(F^*) &= \{\varepsilon\} \end{array}$$

TABLE 2.3 – *Calcul de la Fonction* Null.

$$\begin{array}{rlrl} \mathrm{First}(\emptyset) &= \emptyset & \mathrm{First}(F+G) &= \mathrm{First}(F) \cup \mathrm{First}(G) \\ \mathrm{First}(\varepsilon) &= \emptyset & \mathrm{First}(F \cdot G) &= \begin{cases} \mathrm{First}(F) \cup \mathrm{First}(G) & \text{si } \mathrm{Null}(F) = \{\varepsilon\}, \\ \mathrm{First}(F) & \text{sinon.} \end{cases} \\ \mathrm{First}(a) &= \{a\} & \mathrm{First}(F^*) &= \mathrm{First}(F) \end{array}$$

TABLE 2.4 – *Calcul de la Fonction* First.

$$\begin{array}{rlrl} \mathrm{Last}(\emptyset) &= \emptyset & \mathrm{Last}(F+G) &= \mathrm{Last}(F) \cup \mathrm{Last}(G) \\ \mathrm{Last}(\varepsilon) &= \emptyset & \mathrm{Last}(F \cdot G) &= \begin{cases} \mathrm{Last}(F) \cup \mathrm{Last}(G) & \text{si } \mathrm{Null}(G) = \{\varepsilon\}, \\ \mathrm{Last}(G) & \text{sinon.} \end{cases} \\ \mathrm{Last}(a) &= \{a\} & \mathrm{Last}(F^*) &= \mathrm{Last}(F) \end{array}$$

TABLE 2.5 – *Calcul de la Fonction* Last.

$$\begin{array}{rl} \mathrm{Follow}(\emptyset, x) = \mathrm{Follow}(\varepsilon, x) = \mathrm{Follow}(a, x) &= \emptyset \\ \mathrm{Follow}(F+G, x) &= \mathrm{Follow}(F, x) \cup \mathrm{Follow}(G, x) \\ \mathrm{Follow}(F \cdot G, x) &= \begin{cases} \mathrm{Follow}(F, x) \cup \mathrm{First}(G) & \text{si } x \in \mathrm{Last}(F), \\ \mathrm{Follow}(F, x) \cup \mathrm{Follow}(G, x) & \text{sinon.} \end{cases} \\ \mathrm{Follow}(F^*, x) &= \begin{cases} \mathrm{Follow}(F, x) \cup \mathrm{First}(F) & \text{si } x \in \mathrm{Last}(F), \\ \mathrm{Follow}(F, x) & \text{sinon.} \end{cases} \end{array}$$

TABLE 2.6 – *Calcul de la Fonction* Follow.

Pour les calculs du Follow d'une somme ou d'une concaténation, au moins un des deux ensembles $\mathrm{Follow}(F, x)$ et $\mathrm{Follow}(G, x)$ est vide puisque l'expression E est linéaire.

Tous ces calculs sont équivalents aux définitions données Table 2.1 pour une expression rationnelle E si l'on utilise ces fonctions récursives sur l'expression linéarisée $E^\#$ de E. Cette propriété est trivialement démontrée par induction structurelle sur les expressions rationnelles et leurs langages.

Exemple 2.2 *Reprenons l'expression rationnelle $E = (a+b)^*a + b^*$ de l'Exemple 2.1 et son expression linéarisée $E^\# = (1+2)^*3 + 4^*$. Les fonctions de Glushkov pour l'expression E sont calculables à partir de l'expression rationnelle $E^\#$, et ainsi :*

$$
\begin{aligned}
\mathrm{Pos}(E) &= \mathrm{Pos}(E^{\#}) \\
&= \mathrm{Pos}((1+2)^*3) \cup \mathrm{Pos}(4^*) \\
&= \mathrm{Pos}((1+2)^*) \cup \mathrm{Pos}(3) \\
&\quad \cup\ \mathrm{Pos}(4) \\
&= \mathrm{Pos}(1+2) \cup \{3\} \cup \{4\} \\
&= \mathrm{Pos}(1) \cup \mathrm{Pos}(2) \cup \{3,4\} \\
&= \{1\} \cup \{2\} \cup \{3,4\} \\
&= \{1,2,3,4\}
\end{aligned}
\qquad
\begin{aligned}
\mathrm{Null}(E) &= \mathrm{Null}(E^{\#}) \\
&= \mathrm{Null}((1+2)^*3) \cup \mathrm{Null}(4^*) \\
&= (\mathrm{Null}((1+2)^*) \cap \mathrm{Null}(3)) \\
&\quad \cup\ \{\varepsilon\} \\
&= (\{\varepsilon\} \cap \emptyset) \cup \{\varepsilon\} \\
&= \emptyset \cup \{\varepsilon\} \\
&= \{\varepsilon\}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{First}(E) &= \mathrm{First}(E^{\#}) \\
&= \mathrm{First}((1+2)^*3) \cup \mathrm{First}(4^*) \\
&= \mathrm{First}((1+2)^*) \cup \mathrm{First}(3) \\
&\quad \cup\ \mathrm{First}(4) \\
&= \mathrm{First}(1+2) \cup \{3,4\} \\
&= \mathrm{First}(1) \cup \mathrm{First}(2) \cup \{3,4\} \\
&= \{1,2,3,4\}
\end{aligned}
\qquad
\begin{aligned}
\mathrm{Last}(E) &= \mathrm{Last}(E^{\#}) \\
&= \mathrm{Last}((1+2)^*3) \cup \mathrm{Last}(4^*) \\
&= \mathrm{Last}(3) \cup \mathrm{Last}(4) \\
&= \{3\} \cup \{4\} \\
&= \{3,4\}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Follow}(E,1) &= \mathrm{Follow}(E^{\#},1) \\
&= \mathrm{Follow}((1+2)^*3,1) \\
&= \mathrm{Follow}((1+2)^*,1) \\
&\quad \cup\ \mathrm{First}(3) \\
&= \mathrm{First}(1+2) \cup \{3\} \\
&= \{1,2,3\}
\end{aligned}
\qquad
\begin{aligned}
\mathrm{Follow}(E,2) &= \mathrm{Follow}(E^{\#},2) \\
&= \mathrm{Follow}((1+2)^*3,2) \\
&= \mathrm{Follow}((1+2)^*,2) \\
&\quad \cup\ \mathrm{First}(3) \\
&= \mathrm{First}(1+2) \cup \{3\} \\
&= \{1,2,3\}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Follow}(E,3) &= \mathrm{Follow}(E^{\#},3) \\
&= \mathrm{Follow}((1+2)^*3,3) \\
&= \mathrm{Follow}(3,3) \\
&= \emptyset
\end{aligned}
\qquad
\begin{aligned}
\mathrm{Follow}(E,4) &= \mathrm{Follow}(E^{\#},4) \\
&= \mathrm{Follow}(4^*,4) \\
&= \mathrm{First}(4) \\
&= \{4\}
\end{aligned}
$$

Ces fonctions permettent de calculer l'*automate de Glushkov* ou *automate des positions* de l'expression E reconnaissant le langage dénoté par E. Pour une expression de largeur n, l'automate des positions possède $(n+1)$ états, est standard et homogène et reconnait le langage dénoté par E.

Définition 2.1 *Soit E une expression rationnelle sur un alphabet Σ et soit $A = (\Sigma, Q, I, F, \delta)$ l'automate défini par :*

- $Q = \mathrm{Pos}(E) \cup \{0\}$,
- $I = \{0\}$,
- $F = \begin{cases} \{0\} \cup \mathrm{Last}(E) & \textit{si } \mathrm{Null}(E) = \{\varepsilon\}, \\ \mathrm{Last}(E) & \textit{sinon.} \end{cases}$
- $\delta = \{(q, \mathrm{h}_E(q'), q') \in Q \times \Sigma \times Q \mid q' \in \mathrm{Follow}(E, q)\}$
  $\cup\ \{(0, \mathrm{h}_E(q), q) \in \{0\} \times \Sigma \times Q \mid q \in \mathrm{First}(E)\}$.

L'automate A est appellé automate de Glushkov *ou* automate des positions *de l'expression E.*

Lemme 2.1 *Soit E une expression rationnelle à n lettres et soit A son automate de Glushkov. Alors l'automate A est standard et homogène, possède $(n+1)$ états et reconnait le langage $L(E)$.*

Démonstration. Par construction, l'automate A est standard, homogène et possède $n+1$ états. Montrons alors que $L(E) = L(A)$. Posons $E^{\#}$ l'expres-

sion linéarisée de E.

Soit w un mot de $L(E)$. Si $w = \varepsilon$, par définition $\mathrm{Null}(E) = \{\varepsilon\}$ et ainsi l'état initial de l'automate est terminal, et par conséquent, $w \in L(A)$. Supposons $w = a_1 \cdots a_m$ avec pour tout k de $[\![1, m]\!]$, $a_k \in \Sigma$. Puisque E est une expression simple compatible avec les positions, il existe un mot $w' = x_1 \cdots x_m$ dans $L(E^{\#})$ tel que pour tout k de $[\![1, m]\!]$, $\mathrm{h}(x_k) = a_k$ et tel que $x_1 \in \mathrm{First}(E)$, $x_m \in \mathrm{Last}(E)$, et pour tout k de $[\![2, m]\!]$, $x_k \in \mathrm{Follow}(E, x_{k-1})$. Par construction, $\mathrm{h}(x_1) \in \delta(0, \mathrm{h}(x_1))$, $\mathrm{h}(x_m) \in F$ et pour tout k de $[\![2, m]\!]$, $\mathrm{h}(x_k) \in \delta(\mathrm{h}(x_{k-1}), \mathrm{h}(x_k))$. Ainsi il existe un chemin réussi dans A d'étiquette $\mathrm{h}(w') = w$, et par définition du langage d'un automate, $w \in L(A)$.

Soit w un mot de $L(A)$. Si $w = \varepsilon$, l'état initial 0 est terminal, ce qui est impliqué par $\mathrm{Null}(E) = \{\varepsilon\}$. Ainsi, par définition de la fonction Null, $w \in L(E)$. Supposons $w = a_1 \cdots a_m$ avec pour tout k de $[\![1, m]\!]$, $a_k \in \Sigma$. Par construction, il existe un état x_m de F tel que $\mathrm{x_m} = a_m$ tel qu'il existe un chemin de 0 à l'état x_m. Par construction, il existe $w' = x_1 \cdots x_m$ dans $L(E^{\#})$ tel que pour tout k de $[\![1, m]\!]$, $\mathrm{h}(x_k) = a_k$ et tel que $x_1 \in \mathrm{First}(E)$, $x_m \in \mathrm{Last}(E)$, et pour tout k de $[\![2, m]\!]$, $x_k \in \mathrm{Follow}(E, x_{k-1})$. Puisque E est une expression simple compatible avec les positions, $\mathrm{h}(w') \in L(E)$. Par conséquent, w appartient à $L(E)$. □

Exemple 2.3 *Reprenons l'expression rationnelle $E = (a + b)^*a + b^*$ de l'Exemple 2.1 et ses fonctions de Glushkov :*

$$\begin{array}{rlrl} \mathrm{Pos}(E) &= \{1,2,3,4\} & \mathrm{Follow}(E,1) &= \{1,2,3\} \\ \mathrm{Null}(E) &= \{\varepsilon\} & \mathrm{Follow}(E,2) &= \{1,2,3\} \\ \mathrm{First}(E) &= \{1,2,3,4\} & \mathrm{Follow}(E,3) &= \emptyset \\ \mathrm{Last}(E) &= \{3,4\} & \mathrm{Follow}(E,4) &= \{4\} \end{array}$$

L'automate de Glushkov de l'expression E est l'automate de la Figure 2.1.

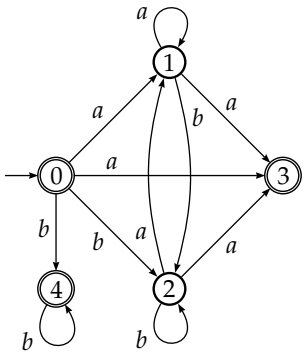


FIGURE 2.1 – *L'Automate de Glushkov de l'Expression $E = (a + b)^*a + b^*$.*

Cette construction s'effectue naïvement en temps $O(n^3)$, où n est la largeur de l'expression de départ. Plusieurs méthodes existent pour construire cet automate en temps quadratique, dues à Brüggemann-Klein [8], Chang et Paige([19] et [20]), ou Champarnaud, Ponty et Ziadi [40].

2.1.2 L'Automate des Follows

La seconde méthode étudiée est la méthode de l'automate des follows, amélioration de l'automate de Glushkov due à Ilie et Yu [28]. La construction de l'automate des follows est une application partielle de la règle de minimisation par la congruence de Myhill-Nerode sur l'ensemble des états d'un DFA (voir Définition 1.10). Elle est basée sur la remarque suivante. Soit p et q deux états d'un DFA $A = (\Sigma, Q, I, F, \delta)$. Si pour toute lettre a de Σ, $\delta(p,a) = \delta(q,a)$, alors les états p et q sont équivalents selon la congruence de Myhill-Nérode. Cette propriété suffisante mais non nécessaire de l'équivalence de deux états permet de définir une congruence plus simple à calculer, afin de déterminer si des états sont fusionnables. Cette congruence s'applique ici sur les états de l'automate de Glushkov de l'expression linéarisée, assimilée aux lettres de l'expression linéarisée. Pour cela, l'état initial 0 de l'automate de Glushkov est assimilé à la position 0. Ainsi, cette position intervient dans les calculs des fonctions de Glushkov. Cette extension à cette nouvelle position redéfinit les fonctions de Glushkov comme suit :

Définition 2.2 *Soit E une expression rationnelle, soit $E^{\#}$ sa linéarisée et soit x une lettre de $\Sigma_{E^{\#}}$. Les extensions à la position 0 des fonctions de Glushkov sont définies comme suit :*

$$\mathrm{Pos}_0(E) = \mathrm{Pos}(E) \cup \{0\}$$

$$\mathrm{Last}_0(E) = \begin{cases} \mathrm{Last}(E) & \text{si } \mathrm{Null}(E) = \emptyset \\ \mathrm{Last}(E) \cup \{0\} & \text{sinon.} \end{cases}$$

$$\mathrm{Follow}_0(E,x) = \begin{cases} \mathrm{First}(E) & \text{si } x = 0 \\ \mathrm{Follow}(E,x) & \text{sinon.} \end{cases}$$

Exemple 2.4 *Considérons l'expression rationnelle $E = (a+b)^*a + b^*$ de l'Exemple 2.1 et son expression linéarisée $E^{\#} = (1+2)^*3 + 4^*$. Les fonctions de Glushkov étendues à la position 0 pour l'expression E sont les suivantes :*

$$\begin{aligned} \mathrm{Pos}_0(E) &= \mathrm{Pos}(E) \cup \{0\} \\ &= \{0,1,2,3,4\} \end{aligned} \qquad \begin{aligned} \mathrm{Last}_0(E) &= \mathrm{Last}(E) \cup \{0\} \\ &= \{0,3,4\} \end{aligned}$$

$$\begin{aligned} \mathrm{Follow}_0(E,1) &= \mathrm{Follow}(E,1) \\ &= \{1,2,3\} \end{aligned} \qquad \begin{aligned} \mathrm{Follow}_0(E,2) &= \mathrm{Follow}(E,2) \\ &= \{1,2,3\} \end{aligned}$$

$$\begin{aligned} \mathrm{Follow}_0(E,3) &= \mathrm{Follow}(E,3) \\ &= \emptyset \end{aligned} \qquad \begin{aligned} \mathrm{Follow}_0(E,4) &= \mathrm{Follow}(E,4) \\ &= \{4\} \end{aligned}$$

$$\begin{aligned} \mathrm{Follow}_0(x,0) &= \mathrm{First}(E) \\ &= \{1,2,3,4\} \end{aligned}$$

Définition 2.3 *Soit E une expression rationnelle sur un alphabet Σ, $E^{\#}$ l'expression linéarisée de E, $A = (\Sigma_{E^{\#}}, Q, I, F, \delta)$ l'automate des positions de $E^{\#}$ et x et y deux états de Q. La relation $\sim_f$ dans Q est définie comme suit :*
$x \sim_f y \Leftrightarrow \big(\mathrm{Follow}_0(E,x) = \mathrm{Follow}_0(E,y)$ *et* $x \in \mathrm{Last}_0(E) \Leftrightarrow y \in \mathrm{Last}_0(E)\big)$.

Lemme 2.2 *La relation $\sim_f$ dans Q est une congruence.*

Démonstration. Montrons que cette relation est reflexive, symétrique, transitive et stable à droite par l'action des symboles. Soit x, y, z trois états de Q et soit a une lettre de $\Sigma_{E^\#}$.

$\sim_f$ **est reflexive :**

$$\text{Follow}_0(E,x) = \text{Follow}_0(E,x) \text{ et } x \in \text{Last}_0(E) \Leftrightarrow x \in \text{Last}_0(E).$$

$\sim_f$ **est symétrique :**

$$(\text{Follow}_0(E,x) = \text{Follow}_0(E,y) \text{ et } x \in \text{Last}_0(E) \Leftrightarrow y \in \text{Last}_0(E))$$
$$\Leftrightarrow (\text{Follow}_0(E,y) = \text{Follow}_0(E,x) \text{ et } y \in \text{Last}_0(E) \Leftrightarrow x \in \text{Last}_0(E))$$

$\sim_f$ **est transitive :**

$$(\text{Follow}_0(E,x) = \text{Follow}_0(E,y) \text{ et } x \in \text{Last}_0(E) \Leftrightarrow y \in \text{Last}_0(E))$$
$$\text{et } (\text{Follow}_0(E,y) = \text{Follow}_0(E,z) \text{ et } y \in \text{Last}_0(E) \Leftrightarrow z \in \text{Last}_0(E))$$
$$\Rightarrow (\text{Follow}_0(E,x) = \text{Follow}_0(E,z) \text{ et } x \in \text{Last}_0(E) \Leftrightarrow z \in \text{Last}_0(E))$$

$\sim_f$ **est stable à droite :**

$$(\text{Follow}_0(E,x) = \text{Follow}_0(E,y) \text{ et } x \in \text{Last}_0(E) \Leftrightarrow y \in \text{Last}_0(E))$$
$$\Rightarrow \delta(x,a) = \delta(y,a) = \{a\} \text{ ou } \delta(x,a) = \delta(y,a) = \emptyset$$
$$\Rightarrow \delta(x,a) \sim_f \delta(y,a)$$

□

En appliquant cette congruence sur les états de l'automate des positions, on peut construire un automate plus petit que l'automate de Glushkov, *l'automate des follows*, défini comme suit :

Définition 2.4 *Soit E une expression rationnelle sur un alphabet Σ. Soit $A = (\Sigma, Q, I, F, \delta)$ l'automate défini par :*

- $Q = \text{Pos}_0(E)/_{\sim_f}$,
- $I = [0]_{\sim_f}$,
- $F = \text{Last}_0(E)/_{\sim_f}$,
- $\delta = \{([q]_{\sim_f}, \text{h}(q'), [q']_{\sim_f}) \in Q \times \Sigma \times Q \mid q' \in \text{Follow}_0(E,q)\}$.

L'automate A est appelé automate des follows de E.

Exemple 2.5 *Considérons l'expression rationnelle $E = (a+b)^*a + b^*$ de l'Exemple 2.1 et ses fonctions de Glushkov étendues à la position 0 :*

$$\text{Pos}_0(E) = \{0,1,2,3,4\} \qquad \text{Last}_0(E) = \{0,3,4\}$$
$$\text{Follow}_0(E,1) = \{1,2,3\} \qquad \text{Follow}_0(E,2) = \{1,2,3\}$$
$$\text{Follow}_0(E,3) = \emptyset \qquad \text{Follow}_0(E,4) = \{4\}$$
$$\text{Follow}_0(E,0) = \{1,2,3,4\}$$

Les états 1 et 2 sont équivalents par $\sim_f$. L'automate des follows de l'expression E est l'automate A de la Figure 2.2.

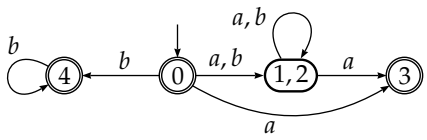


FIGURE 2.2 – *L'Automate des Follows de l'Expression $E = (a+b)^*a + b^*$.*

En fait, cet algorithme réalise la minimisation de l'automate de Glushkov de l'expression linéarisée. Cependant, de nouvelles fusions peuvent être réalisées après délinéarisation, comme le montre l'Exemple 2.6.

Exemple 2.6 *Considérons l'expression rationnelle $E = (b + c)a^* + da^*$ et son expression linéarisée $E^{\#} = (1 + 2)3^* + 45^*$. L'automate de Glushkov de E est représenté Figure 2.3. L'automate des follows de E est représenté Figure 2.4. En appliquant la fusion d'états par l'équivalence de Myhill-Nerode, on obtient alors l'automate de la Figure 2.5 par fusion des états $\{1, 2, 3\}$ avec $\{4, 5\}$.*

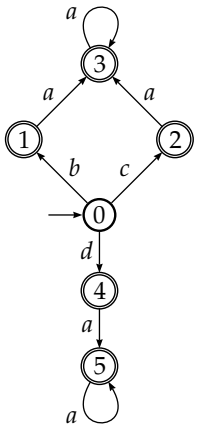


FIGURE 2.3 – *Automate de Glushkov de E.*

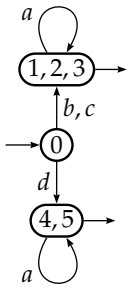


FIGURE 2.4 – *Automate des Follows de E.*

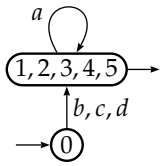


FIGURE 2.5 – *Automate Minimal de $L(E)$.*

2.1.3 L'Automate des Dérivées

La troisième méthode présentée est la dérivation des expressions rationnelles, due à Brzozowski ([11]). Cette méthode calcule un automate déterministe par une construction proche de celle par quotient de langage (voir la Définition 1.9 dans la sous-section 1.4.4). Cette approche porte sur les expressions rationnelles, et bien que l'on réalise un quotient, l'opération appliquée aux expressions rationnelles est appelée *dérivation*. On montrera alors que le langage dénoté par une expression dérivée est le même que le langage obtenu par quotient du langage de départ. Cette méthode produit un automate déterministe. Par conséquent, une explosion du nombre d'états par rapport au nombre de lettres de l'expression peut avoir lieu.

Les formules de dérivation d'une expression rationnelle par rapport à une lettre, s'appliquant inductivement sur la structure de l'expression rationnelle sont définies comme suit :

Définition 2.5 *Soit E une expression rationnelle sur un alphabet Σ et soit a et b deux lettres de Σ distinctes. La* dérivée de *E* par rapport à *a est l'expression notée $\frac{d}{d_a}(E)$ et est calculée inductivement comme suit :*

$$\frac{d}{d_a}(\emptyset) = \frac{d}{d_a}(\varepsilon) = \frac{d}{d_a}(b) = \emptyset,\ \frac{d}{d_a}(a) = \varepsilon,$$
$$\frac{d}{d_a}(F+G) = \frac{d}{d_a}(F) + \frac{d}{d_a}(G),\ \frac{d}{d_a}(F^*) = \frac{d}{d_a}(F)\cdot F^*,$$
$$\frac{d}{d_a}(F\cdot G) = \begin{cases} \frac{d}{d_a}(F)\cdot G + \frac{d}{d_a}(G) & \textit{si } \text{Null}(F) = \{\varepsilon\}, \\ \frac{d}{d_a}(F)\cdot G & \textit{sinon.} \end{cases}$$

Cette notion de dérivation s'étend très facilement à une dérivation par rapport à un mot w. En effet, pour E une expression rationnelle sur un alphabet Σ et w un mot de Σ^*, on pose :

$$\frac{d}{d_w}(E) = \begin{cases} \frac{d}{d_{w'}}(\frac{d}{d_a}(E)) & \text{si } w = a\cdot w' \text{ avec } a\in\Sigma \text{ et } w'\in\Sigma^* \\ E & \text{si } w = \varepsilon. \end{cases}$$

Remarquons alors que calculer le quotient d'un langage dénoté par une expression E par une lettre a revient à calculer le langage dénoté par la dérivée de E par la lettre a :

$$a^{-1}(L(E)) = L(\tfrac{d}{d_a}(E)).$$

Cette propriété s'étend aux mots de Σ^* :

Proposition 2.1 (Brzozowski, [11]) *Soit E une expression rationnelle simple sur un alphabet Σ et w un mot de Σ^*. Alors :*

$$w^{-1}(L(E)) = L(\tfrac{d}{d_w}(E)).$$

Toute expression E' obtenue par dérivation d'une expression E par rapport à un mot w de Σ^* est appelée *dérivée de E*. Puisque le mot vide appartient à Σ^* et que $\frac{d}{d_\varepsilon}(E) = E$, l'expression E est également une dérivée de E. À partir de ces formules de dérivation, on peut alors calculer l'ensemble des dérivées d'une expression rationnelle. Mais un problème se pose. En effet, cet ensemble de dérivées est-il fini ou infini ?

Exemple 2.7 *Soit $E = a^*(a+b)^*$ une expression rationnelle. Calculons les dérivées par rapport aux mots sur l'alphabet $\{a,b\}$.*

$\frac{d}{d_\varepsilon}(E) = E$

$$\begin{aligned}\frac{d}{d_a}(E) &= \frac{d}{d_a}(a^*)\cdot(a+b)^* + \frac{d}{d_a}((a+b)^*) \\ &= \frac{d}{d_a}(a)\cdot a^*\cdot(a+b)^* + \frac{d}{d_a}(a+b)\cdot(a+b)^* \\ &= a^*\cdot(a+b)^* + (a+b)^* \\ &= E + (a+b)^*\end{aligned}$$

$$\begin{aligned}\frac{d}{d_{aa}}(E) &= \frac{d}{d_a}(\frac{d}{d_a}(E)) \\ &= \frac{d}{d_a}(E + (a+b)^*) \\ &= \frac{d}{d_a}(E) + \frac{d}{d_a}((a+b)^*) \\ &= E + (a+b)^* + (a+b)^*\end{aligned}$$

$$\begin{aligned}\frac{d}{d_{aaa}}(E) &= \frac{d}{d_a}(\frac{d}{d_{aa}}(E)) \\ &= \frac{d}{d_a}(E + (a+b)^* + (a+b)^*) \\ &= E + (a+b)^* + (a+b)^* + (a+b)^*\end{aligned}$$

On peut alors en déduire la propriété suivante :

$$\frac{d}{d_{a^n}}(E) = \begin{cases} E & \textit{si } n = 0 \\ \frac{d}{d_{a^{n-1}}}(E) + (a+b)^* & \textit{sinon.} \end{cases}$$

L'ensemble des dérivées de l'expression E est donc infini, puisque chacune des dérivations successives par la lettre a produit une expression rationnelle syntaxiquement différente des autres.

Ainsi, l'ensemble des dérivées d'une expression peut être infini. Cependant, cette propriété n'est plus vérifiée si l'on considère le quotient d'expression rationnelle selon trois règles de transformation syntaxique (*associativité*, *commutativité* et *idempotence*) définies comme suit :

(A)ssociativité : $(E+F)+G \sim_{\mathbf{A}} E+(F+G)$,
(C)ommutativité : $E+F \sim_{\mathbf{C}} F+E$,
(I)dempotence : $E+E \sim_{\mathbf{I}} E$.

Théorème 2.1 (Brzozowski, [11]) *Soit E une expression rationnelle sur un alphabet Σ et soit $\mathcal{D}_E$ l'ensemble de ses dérivées par rapport aux mots de Σ^*. Alors $\mathcal{D}_E$ est fini modulo les réductions **(ACI)**.*

Dans la suite, par souci de concision, nous assimilerons l'ensemble $\mathcal{D}_E$ à sa réduction modulo ACI.

Exemple 2.8 *Reprenons l'expression rationnelle $E = a^*(a+b)^*$ de l'Exemple 2.7 et calculons l'ensemble de ses dérivées $\mathcal{D}_E$:*

$E_0 = \frac{d}{d_\varepsilon}(E) = E$

$$\begin{aligned} E_1 &= \tfrac{d}{d_a}(E) \\ &= \tfrac{d}{d_a}(a^*(a+b)^*) \\ &= \tfrac{d}{d_a}(a^*)(a+b)^* + \tfrac{d}{d_a}((a+b)^*) \\ &= a^* \cdot (a+b)^* + (a+b)^* \\ &= E + (a+b)^* \end{aligned} \qquad \begin{aligned} E_2 &= \tfrac{d}{d_b}(E) \\ &= \tfrac{d}{d_b}(a^*(a+b)^*) \\ &= \tfrac{d}{d_b}(a^*)(a+b)^* + \tfrac{d}{d_b}((a+b)^*) \\ &= \tfrac{d}{d_b}(a)a^*(a+b)^* + \tfrac{d}{d_b}(a+b)(a+b)^* \\ &= (a+b)^* \end{aligned}$$

$$\begin{aligned} E_3 &= \tfrac{d}{d_{aa}}(E) \\ &= \tfrac{d}{d_a}(\tfrac{d}{d_a}(E)) \\ &= \tfrac{d}{d_a}(E + (a+b)^*) \\ &= \tfrac{d}{d_a}(E) + \tfrac{d}{d_a}((a+b)^*) \\ &= E + (a+b)^* + (a+b)^* \\ &\sim_{ACI} E_1 \end{aligned} \qquad \begin{aligned} E_4 &= \tfrac{d}{d_{ab}}(E) \\ &= \tfrac{d}{d_b}(\tfrac{d}{d_a}(E)) \\ &= \tfrac{d}{d_b}(a^* \cdot (a+b)^* + (a+b)^*) \\ &= \tfrac{d}{d_b}(a^* \cdot (a+b)^*) + \tfrac{d}{d_b}((a+b)^*) \\ &= \tfrac{d}{d_b}((a+b)^*) + \tfrac{d}{d_b}((a+b)^*) \\ &= (a+b)^* + (a+b)^* \\ &\sim_{ACI} E_2 \end{aligned}$$

$$\begin{aligned} E_4 &= \tfrac{d}{d_{ba}}(E) \\ &= \tfrac{d}{d_a}(\tfrac{d}{d_b}(E)) \\ &= \tfrac{d}{d_a}((a+b)^*) \\ &= \tfrac{d}{d_a}(a+b)(a+b)^* \\ &= (a+b)^* \\ &= E_2 \end{aligned} \qquad \begin{aligned} E_5 &= \tfrac{d}{d_{bb}}(E) \\ &= \tfrac{d}{d_b}(\tfrac{d}{d_b}(E)) \\ &= \tfrac{d}{d_b}((a+b)^*) \\ &= \tfrac{d}{d_b}(a+b)(a+b)^* \\ &= (a+b)^* \\ &= E_2 \end{aligned}$$

L'ensemble des dérivées de E est donc $\mathcal{D}_E = \{E, E_1, E_2\}$.

Le fait d'obtenir un ensemble fini de dérivées permet de calculer un automate déterministe reconnaissant le langage dénoté par une expression rationnelle.

Définition 2.6 *Soit E une expression rationnelle sur un alphabet Σ et soit $\mathcal{D}_E$ l'ensemble de ses dérivées. Soit $A = (\Sigma, Q, I, F, \delta)$ l'automate défini par :*

- $Q = \mathcal{D}_E$,
- $I = \{E\}$,
- $F = \{q \in Q \mid \varepsilon \in L(q)\}$,
- $\delta = \{(q, a, q') \in Q \times \Sigma \times Q \mid \frac{d}{d_a}(q) = q'\}$.

L'automate A est appelé automate des expressions dérivées *de E.*

Lemme 2.3 *Soit E une expression rationnelle sur un alphabet Σ et soit A son automate des expressions dérivées. Alors A est un DFA reconnaissant le langage dénoté par E.*

Démonstration. Par construction, A est un DFA, puisqu'il possède un unique état initial et que chacun de ses états possède au plus une transition sortante par lettre de l'alphabet. De plus, puisque le langage droit de l'état initial E est dénoté par E, et qu'il correspond à la dérivation de E par le mot vide, par construction de A on a alors pour tout mot w de Σ^* :

$$w \in L(E) \Leftrightarrow w \in L(A).$$

□

Exemple 2.9 *Reprenons l'expression rationnelle $E = a^*(a+b)^*$ de l'Exemple 2.7 et l'ensemble de ses dérivées $\mathcal{D}_E = \{E, E_1, E_2\}$ avec $E_1 = E + (a+b)^*$ et $E_2 = (a+b)^*$. L'automate des expressions dérivées de E est l'automate A de la Figure 2.6.*

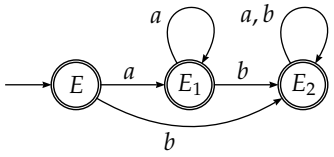


FIGURE 2.6 – *L'Automate des Expressions Dérivées de l'Expression $a^*(a+b)^*$.*

2.1.4 L'Automate des Dérivées Partielles

La quatrième méthode de conversion d'une expression rationnelle vers un NFA est l'utilisation des dérivées partielles, due à Antimirov [2], et appliquant un principe relativement proche de la dérivation, mais où le résultat de la dérivation n'est plus une expression, mais un ensemble d'expressions. Cela conduit à la construction non plus d'un DFA, mais d'un NFA à possédant au maximum $(n+1)$ états, à partir d'une expression à n lettres. Cette perte de déterminisme permet d'éviter les explosions du nombre d'états pouvant avoir lieu dans la méthode par dérivation. Trente ans plus tôt, Mirkin, dans [36], définissait des bases et prébases pour les expressions rationnelles par le biais d'équations. Champarnaud et Ziadi ont montré le lien entre les prébases de Mirkin et les dérivées partielles d'Antimirov dans [17]. Pour cela, l'automate des dérivées partielles est également appelé automate des équations.

Les opérations de dérivation partielle sont définies comme suit :

Définition 2.7 *Soit E une expression rationnelle sur un alphabet Σ et soit a et b deux lettres de Σ distinctes. La dérivée partielle de E par rapport à a est un ensemble d'expressions noté $\frac{\partial}{\partial_a}(E)$ et calculé inductivement comme suit :*

$$\frac{\partial}{\partial_a}(\emptyset) = \frac{\partial}{\partial_a}(\varepsilon) = \frac{\partial}{\partial_a}(b) = \emptyset,\ \frac{\partial}{\partial_a}(a) = \{\varepsilon\},$$
$$\frac{\partial}{\partial_a}(F+G) = \frac{\partial}{\partial_a}(F) \cup \frac{\partial}{\partial_a}(G),\ \frac{\partial}{\partial_a}(F^*) = \frac{\partial}{\partial_a}(F)\cdot F^*,$$
$$\frac{\partial}{\partial_a}(F\cdot G) = \begin{cases} \frac{\partial}{\partial_a}(F)\cdot G \cup \frac{\partial}{\partial_a}(G) & \textit{si } \mathrm{Null}(F) = \{\varepsilon\}, \\ \frac{\partial}{\partial_a}(F)\cdot G & \textit{sinon.} \end{cases}$$

Les formules de dérivation pour la concaténation et l'étoile de Kleene décrivent une concaténation entre un ensemble d'expressions rationnelles et une expression rationnelle. Soit $\mathcal{E}$ un ensemble fini d'expressions et soit F une expression rationnelle. Alors :

$$\mathcal{E}\cdot F = \bigcup_{E\in\mathcal{E}} E\cdot F.$$

De plus, les formules de dérivation partielle interviennent également sur des ensembles d'expressions rationnelles. Ainsi, en considérant $\mathcal{E}$ un ensemble d'expressions et a une lettre de l'alphabet,

$$\frac{\partial}{\partial_a}(\mathcal{E}) = \bigcup_{E\in\mathcal{E}} \frac{\partial}{\partial_a}(E).$$

Enfin, l'extension de la dérivation partielle d'une expression sur un alphabet Σ par rapport aux mots de Σ^* se réalise de la même manière que pour la dérivation. Soit E une expression rationnelle sur un alphabet Σ, et soit w un mot de Σ^*. La dérivation partielle de E par rapport à w se calcule comme suit :

$$\frac{\partial}{\partial_w}(E) = \begin{cases} \frac{\partial}{\partial_{w'}}(\frac{\partial}{\partial_a}(E)) & \text{si } w = a\cdot w' \text{ avec } a\in\Sigma \text{ et } w'\in\Sigma^*, \\ \frac{\partial}{\partial_w}(E) = \{E\} & \text{si } w = \varepsilon. \end{cases}$$

Soit E une expression rationnelle sur un alphabet Σ. Toute expression E' contenue dans un ensemble d'expressions obtenu à partir d'une dérivation partielle de E par rapport à un mot w de Σ^* est appelée *dérivée partielle de E* (ou *terme dérivé* de E selon Sakarovitch [43]). Puisque le mot vide est un mot de Σ^*, l'expression E est une dérivée partielle de E. On peut alors considérer l'ensemble des dérivées partielles d'une expression. Cet ensemble n'est pas quotienté par les règles de réduction **ACI**, mais uniquement par les identités triviales (voir Définition 1.15). Et l'ensemble des dérivées partielles obtenu est non seulement fini, mais son cardinal est aussi inférieur ou égal à $(n+1)$, où n est le nombre de lettres de l'expression de départ :

Théorème 2.2 (Antimirov, [2]) *Soit E une expression rationnelle à n lettres sur un alphabet Σ. Soit $\mathcal{D}'_E$ l'ensemble des dérivées partielles de E par rapport aux mots de Σ^*. Alors :*

$$\#\mathcal{D}'_E \leq n+1.$$

Exemple 2.10 *Soit $E = ba^*b + ba^*(b+c)$ une expression rationnelle. Les dérivées partielles de E se calculent comme suit :*

$$\frac{\partial}{\partial_\varepsilon}(E) = E,\quad \frac{\partial}{\partial_a}(E) = \frac{\partial}{\partial_c}(E) = \emptyset,\quad \frac{\partial}{\partial_b}(E) = \{a^*b, a^*(b+c)\}$$
$$\frac{\partial}{\partial_a}(a^*b) = \{a^*b\},\quad \frac{\partial}{\partial_b}(a^*b) = \{\varepsilon\},\quad \frac{\partial}{\partial_c}(a^*b) = \emptyset,$$

$\frac{\partial}{\partial_a}(a^*(b+c)) = \{a^*(b+c)\}$, $\frac{\partial}{\partial_b}(a^*(b+c)) = \{\varepsilon\}$, $\frac{\partial}{\partial_c}(a^*(b+c)) = \{\varepsilon\}$
Ainsi, l'ensemble des dérivées partielles de E est $\mathcal{D}'_E = \{\varepsilon, a^*b, a^*(b+c), E\}$.

Cet ensemble fini de dérivées partielles permet la construction d'un NFA reconnaissant le langage dénoté par une expression rationnelle.

Définition 2.8 *Soit E une expression rationnelle sur un alphabet* Σ *et soit* $\mathcal{D}'_E$ *l'ensemble des dérivées partielles de E. Soit* $A = (\Sigma, Q, I, T, \delta)$ *l'automate défini par :*
- $Q = \mathcal{D}'_E$,
- $I = \{E\}$,
- $F = \{q \in Q \mid \varepsilon \in L(q)\}$,
- $\delta = \{(q, a, q') \in Q \times \Sigma \times Q \mid q' \in \frac{\partial}{\partial_a}(q)\}$.

L'automate A est appelé automate des dérivées partielles*de l'expression rationnelle E.*

Lemme 2.4 *Soit E une expression rationnelle à n lettres et soit A son automate des dérivées partielles. Alors* $L(A) = L(E)$ *et l'automate A possède au plus* $(n+1)$ *états.*

Démonstration. Par construction, l'automate A possède au plus $(n+1)$ états selon le Théorème 2.2. De plus, puisque le langage droit de l'état initial E est dénoté par E, et qu'il correspond à la dérivation partielle de E par le mot vide, par construction de A on a alors pour tout mot w de Σ^* :

$$w \in L(E) \Leftrightarrow w \in L(A).$$

□

Exemple 2.11 *Reprenons l'expression* $E = ba^*b + ba^*(b+c)$ *de l'Exemple 2.10 et ses dérivées partielles* $\mathcal{D}'_E = \{\varepsilon, a^*b, a^*(b+c), E\}$. *L'automate des dérivées partielles de E est l'automate de la Figure 2.7.*

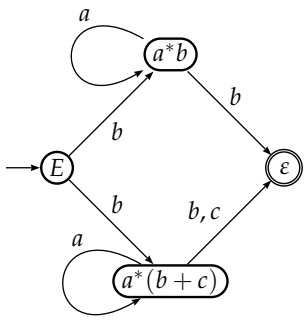


FIGURE 2.7 – *L'Automate des Dérivées Partielles de l'Expression E.*

2.1.5 Automate des C-Continuations

La dernière méthode de conversion d'une expression rationnelle E vers un NFA présentée est celle des continuations, introduite par Berry et Sethi dans [4] et raffinée par Champarnaud et Ziadi dans [16], utilisant la notion

de dérivation de Brzozowski, mais en l'appliquant sur l'expression linéarisée $E^{\#}$. Bien que l'automate des dérivées de l'expression $E^{\#}$ soit déterministe, cette propriété est perdue lors de la délinéarisation de l'automate, et l'automate produit est alors plus petit que l'automate des dérivées de E (jusqu'à un rapport exponentiel du nombre des états). En fait, l'automate calculé par cette méthode, appelé automate des c-continuations, est isomorphe à l'automate des positions. Les c-continuations sont calculées à partir des c-dérivées définies comme suit :

Définition 2.9 (Champarnaud et Ziadi [16]) *Soit E une expression rationnelle sur un alphabet Σ et soit a et b deux lettres de Σ distinctes. La* c-dérivée de E par rapport à a *est l'expression notée $d_a(E)$ et calculée inductivement comme suit :*

$$d_a(\emptyset) = d_a(b) = d_a(\varepsilon) = \emptyset \qquad d_a(a) = \varepsilon$$

$$d_a(F+G) = \begin{cases} d_a(F) \text{ si } d_a(F) \neq \emptyset, \\ d_a(G) \text{ sinon.} \end{cases} \qquad d_a(F \cdot G) = \begin{cases} d_a(F) \cdot G & \text{si } d_a(F) \neq \emptyset, \\ \mathrm{Null}(F) \cdot d_a(G) & \text{sinon.} \end{cases}$$

$$d_a(F^*) = d_a(F) \cdot F^*$$

Cette notion de c-dérivation d'une expression E sur un alphabet Σ par rapport à une lettre de Σ s'étend très facilement à une c-dérivation par rapport à un mot w de Σ^*. En effet, pour E une expression rationnelle et $w = a \cdot w'$ un mot, on a :

$$d_w(E) = \begin{cases} d_{w'}(d_a(E)) & \text{si } w = a \cdot w' \text{ avec } a \in \Sigma \text{ et } w' \in \Sigma^* \\ d_w(E) = E & \text{si } w = \varepsilon. \end{cases}$$

De plus, le langage dénoté par une c-dérivée d'une expression linéaire E par rapport à un mot de Σ^* est égal au quotient de $L(E)$ par w.

Proposition 2.2 (Champarnaud et Ziadi [16]) *Soit E une expression linéaire sur un alphabet Σ et w un mot de Σ^*. Alors :*

$$L(d_w(E)) = w^{-1}(L(E)).$$

Le théorème suivant permet de définir une c-dérivation canonique à partir d'une expression linéaire sur un alphabet Σ pour un symbole de Σ donné :

Théorème 2.3 (Champarnaud et Ziadi [16]) *Soit E une expression linéaire sur un alphabet Σ. Soit a un symbole de Σ. Pour tout mot u, v de Σ^*, la proposition suivante est vérifiée :*

$$(d_{ua}(E) \neq \emptyset \wedge d_{va}(E) \neq \emptyset) \Rightarrow d_{ua}(E) = d_{va}(E).$$

L'unicité de ces c-dérivations canoniques pour un symbole donné permet de définir la *c-continuation*. Ainsi, pour E une expression linéaire sur un alphabet Σ et a une lettre de Σ, la c-continuation de E par rapport à a, notée $c_a(E)$ est l'unique valeur des c-dérivées non-nulles $d_{ua}(E)$.

Proposition 2.3 (Champarnaud et Ziadi [16]) *Soit E une expression rationnelle linéaire sur un alphabet Σ. Soit a une lettre de Σ. La c-continuation de E par rapport à a, notée*

c_a est calculée comme suit :

$$c_a(a) = \varepsilon \qquad c_a(F^*) = c_a(F) \cdot F^*$$
$$c_a(F+G) = \begin{cases} c_a(F) \text{ si } c_a(F) \neq \emptyset, \\ c_a(G) \text{ sinon.} \end{cases} \qquad c_a(F \cdot G) = \begin{cases} c_a(F) \cdot G \text{ si } c_a(F) \neq \emptyset, \\ c_a(G) \quad \text{sinon.} \end{cases}$$

Exemple 2.12 *Considérons l'expression $E = (a+b)^*a + b^*$ ainsi que son expression linéarisée $E^\# = (1+2)^*3 + 4^*$. Les c-continuations de $E^\#$ sont calculées comme suit :*

$$\begin{aligned} c_1(E) &= c_1((1+2)^*3) & c_2(E) &= c_2((1+2)^*3) \\ &= c_1((1+2)^*)3 & &= c_2((1+2)^*)3 \\ &= c_1(1+2)(1+2)^*3 & &= c_2(1+2)(1+2)^*3 \\ &= c_1(1)(1+2)^*3 & &= c_2(2)(1+2)^*3 \\ &= (1+2)^*3 & &= (1+2)^*3 \end{aligned}$$

$$\begin{aligned} c_3(E) &= c_3(3) & c_4(E) &= c_4(4^*) \\ &= \varepsilon & &= c_4(4)4^* \\ & & &= 4^* \end{aligned}$$

L'ensemble des c-continuations de $E^\#$ est donc
$\mathcal{C}_{E^\#} = \{c_1(E) = (1+2)^*3, c_2(E) = (1+2)^*3, c_3(E) = \varepsilon, c_4(E) = 4^*\}$.

L'ensemble des c-continuations de l'expression linéarisée permet de calculer un NFA isomorphe à l'automate des positions :

Définition 2.10 *Soit E une expression rationnelle sur un alphabet Σ, soit $E^\#$ son expression linéarisée et soit $\mathcal{C}_{E^\#} = \{c_x \mid x \in \Sigma_{E^\#}\}$ l'ensemble des c-continuations de $E^\#$. Par convention, on pose $c_\varepsilon = E^\#$. Soit $A = (\Sigma, Q, I, F, \delta)$ l'automate défini par :*

- $Q = \{(x, c_x) \in \Sigma_{E^\#} \times \mathcal{C}_{E^\#}\} \cup \{(\varepsilon, c_\varepsilon)\}$,
- $I = \{(\varepsilon, c_\varepsilon)\}$,
- $F = \{(x, c_x) \in Q \mid \varepsilon \in L(c_x)\}$,
- $\delta = \{(x, c_x), \mathrm{h}(y), (y, c_y) \in Q \times \Sigma \times Q \mid d_y(c_x) = c_y\}$.

L'automate A est appelé automate des c-continuations *de l'expression rationnelle E.*

Théorème 2.4 (Champarnaud et Ziadi [16]) *Soit E une expression rationnelle. L'automate des positions de E et l'automate des c-continuations de E sont isomorphes.*

Corollaire 2.1 *Soit E une expression rationelle. L'automate des c-continuations de E reconnait le langage $L(E)$.*

Exemple 2.13 *Reprenons l'expression $E = (a+b)^*a + b^*$ de l'Exemple 2.12 et l'ensemble des c-continuations de son expression linéarisée $E^\# = (1+2)^*3 + 4^*$. L'ensemble des c-continuations de $E^\#$ est*

$\mathcal{C}_{E^\#} = \{c_1(E) = (1+2)^*3, c_2(E) = (1+2)^*3, c_3(E) = \varepsilon, c_4(E) = 4^*\}$.

L'automate des c-continuations de E est représenté Figure 2.8. Remarquons que l'automate des c-continuations de E est isomorphe à l'automate des positions de E, représenté Figure 2.1.

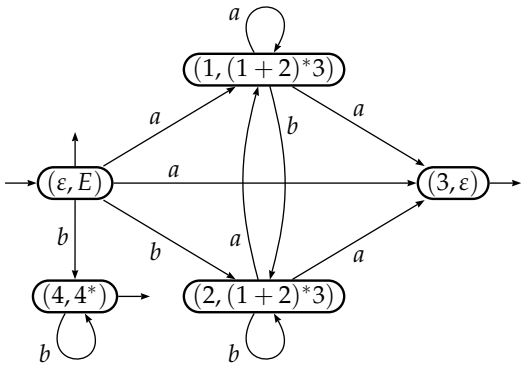


FIGURE 2.8 – *L'Automate des C-Continuations de l'Expression* $E = (a + b)^*a + b^*$.

2.1.6 Relations entre les Automates Produits par ces Méthodes

Selon le Théorème 2.4, les automates des positions et des c-continuations sont isomorphes. D'autres liens existent entre les différentes méthodes de construction présentées. Par exemple, Champarnaud et Ziadi ont montré dans [16] que l'automate des dérivées partielles est un quotient de l'automate des c-continuations par la relation d'équivalence définie comme suit.

Définition 2.11 (Champarnaud et Ziadi [16]) *Soit E une expression rationnelle sur un alphabet Σ et soit $A = (\Sigma, Q, I, F, \delta)$ l'automate des c-continuations de E. La relation $\sim_{\mathrm{h}_E}$ est défini dans Q par :*

$$(x, c_x) \sim_{\mathrm{h}_E} (y, c_y) \Leftrightarrow \mathrm{h}_E(c_x) = \mathrm{h}_E(c_y).$$

Lemme 2.5 (Champarnaud et Ziadi [16]) *Soit E une expression rationnelle sur un alphabet Σ et soit $A = (\Sigma, Q, I, F, \delta)$ l'automate des c-continuations de E. La relation $\sim_{\mathrm{h}_E}$ est une congruence dans Q.*

Théorème 2.5 (Champarnaud et Ziadi [16]) *Soit E une expression rationnelle, soit A son automate des c-continuations et soit A' son automate des dérivées partielles. Alors A' et $A/_{\sim_{\mathrm{h}_E}}$ sont isomorphes.*

Deux autres constructions sont également liées, et cela par l'algorithme de déterminisation (Algorithme 1.5) :

Théorème 2.6 (Antimirov [2]) *Soit E une expression rationnelle sur un alphabet Σ et w un mot de Σ^*. Alors :*

$$\tfrac{d}{d_w}(E) \sim_{ACI} \textstyle\sum_{E' \in \frac{\partial}{\partial_w}(E)} E'.$$

Corollaire 2.2 *Soit E une expression rationnelle, soit A son automate des dérivées partielles, soit A' son automate des dérivées, et soit A'' le déterminisé de A selon l'Algorithme 1.5. Alors A'et A'' sont isomorphes.*

Le schéma représenté Figure 2.9 récapitule les différents liens énoncés.

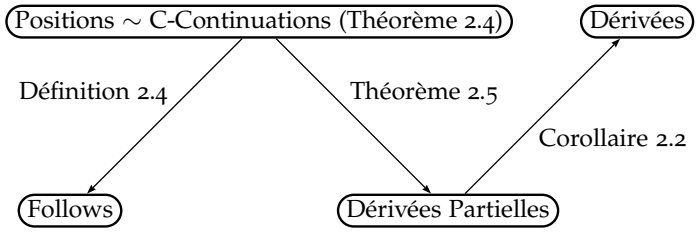


FIGURE 2.9 – *Récapitulatif des Relations entre les Différentes Constructions.*

2.2 Convertir un NFA en une Expression Rationnelle

La section précécente démontre l'inclusion de l'ensemble des langages rationnels dans celui des langages reconnaissables. Cette section présente différentes méthodes de construction d'une expression rationnelle dénotant le langage reconnu par un NFA et ainsi démontre l'inclusion de l'ensemble des langages reconnaissables par automate dans celui des langages rationnels. Pour cela, quatre méthodes transformant un NFA en une expression rationnelle sont présentées.

2.2.1 Méthode par Résolution de Système d'Équations

La première méthode est celle de résolution de système d'équations, introduite par Arden dans [3]. À partir d'un automate donné, la première étape calcule différentes équations sur les états de l'automate, liant les différents langages droits des états de l'automate entre eux. Or, selon le Corollaire 1.1, calculer l'union des expressions rationnelles dénotant les langages droits des états initiaux revient à calculer une expression rationnelle dénotant le langage de l'automate. Ainsi, considérons le système d'équations d'expressions rationnelles suivant, noté $\mathcal{E}_A$, pour $A = (\Sigma, Q, I, F, \delta)$ un NFA, et p et q deux états de Q :

$$\begin{aligned} X_p &= \Sigma_{q\in Q}\delta_{p,q}\cdot X_q + \mathrm{Final}(p) \\ \delta_{p,q} &= \Sigma_{a\in\Sigma | q\in\delta(p,a)} a \\ \mathrm{Final}(p) &= \begin{cases} \varepsilon & \text{si } p \in F, \\ \emptyset & \text{sinon.} \end{cases} \end{aligned}$$

Exemple 2.14 *Soit A le NFA de la Figure 2.10. Le système d'équations associé à A est le système $\mathcal{E}_A$ suivant :*

$$\begin{aligned} X_0 &= & a\cdot X_1 & & + \varepsilon \\ X_1 &= & b\cdot X_1 &+ b\cdot X_2 + \emptyset \\ X_2 &= a\cdot X_0 & &+ a\cdot X_2 + \varepsilon \end{aligned}$$

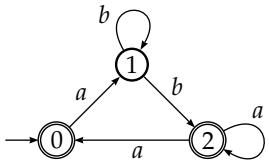


FIGURE 2.10 – *L'Automate A.*

Soit A un automate, Q son ensemble d'états et $\mathcal{E}_A$ le système d'équations de A. Par construction, pour q un état de Q, toute expression X_q de $\mathcal{E}_A$ dénote le langage droit de q dans A. Le lemme suivant, appelé par la suite Lemme d'Arden est utilisé afin de résoudre $\mathcal{E}_A$.

Lemme 2.6 (Arden [3]) *Soit L_1 et L_2 deux langages. Si $\varepsilon \notin L_1$ alors le langage $L_1^* \cdot L_2$ est l'unique solution de l'équation $X = L_1 \cdot X \cup L_2$.*

Par extension, l'expression rationnelle $E_1^* \cdot E_2$ est une solution de l'équation $X = E_1 \cdot X + E_2$, mais cette solution n'est pas unique (toute expression équivalente dénotant $L(E_1^* \cdot E_2)$ pourrait être solution).

Ainsi, pour résoudre un système d'équations $\mathcal{E}_A$, il suffit d'appliquer une suite d'élimination/remplacement de variable et d'application du Lemme d'Arden.

Exemple 2.15 *Considérons le système $\mathcal{E}_A$ de l'Exemple 2.14 :*

$$\begin{aligned} X_0 &= a \cdot X_1 + \varepsilon \\ X_1 &= b \cdot X_1 + b \cdot X_2 + \emptyset \\ X_2 &= a \cdot X_0 + a \cdot X_2 + \varepsilon \end{aligned}$$

On peut résoudre ce système comme suit :

(Remplacement de X_0 dans X_2 :)
$$\begin{aligned} X_2 &= aX_0 + aX_2 + \varepsilon \\ &= a(aX_1 + \varepsilon) + aX_2 + \varepsilon \\ &= aX_2 + a^2X_1 + a + \varepsilon \end{aligned}$$

(Lemme d'Arden sur X_2 :)
$$X_2 = a^*(a^2X_1 + a + \varepsilon)$$

(Remplacement de X_2 dans X_1 :)
$$\begin{aligned} X_1 &= bX_1 + bX_2 + \emptyset \\ &= bX_1 + ba^*(a^2X_1 + a + \varepsilon) \\ &= (b + ba^*a^2)X_1 + ba^*a + ba^* \\ &= (b + ba^*a^2)X_1 + ba^* \end{aligned}$$

(Lemme d'Arden sur X_1 :)
$$X_1 = (b + ba^*a^2)^*ba^*$$

(Remplacement de X_1 dans X_0 :)
$$\begin{aligned} X_0 &= aX_1 + \varepsilon \\ &= a(b + ba^*a^2)^*ba^* + \varepsilon \end{aligned}$$

(Remplacement de X_1 dans X_2 :)
$$\begin{aligned} X_2 &= a^*(a^2X_1 + \varepsilon) \\ &= a^*(a^2((b + ba^*a^2)^*ba^*) + \varepsilon) \end{aligned}$$

Et puisque l'automate A de la Figure 2.10 possède un unique état initial, l'expression $E_0 = a(b + ba^*a^2)^*ba^* + \varepsilon$ dénotant le langage droit de l'état 0

dénote le langage de l'automate A.

2.2.2 Méthode MNY

La seconde méthode est l'algorithme de Mac-Naughton et Yamada (MNY), présenté dans [35] et basé sur un calcul récursif de langages et d'expressions de sous-automates. Le principe est simple : on suppose les états ordonnés, et on construit le langage reconnu (ainsi que des expressions le dénotant) par un sous-automate de l'automate de départ. À partir de ce langage, on calcule le langage d'un automate plus grand jusqu'à obtenir l'automate de départ.

Considérons un automate $A = (\Sigma, Q, I, F, \delta)$. On peut décomposer l'automate A comme une union d'automates possédant un unique état initial et un unique état terminal. Comme le montre l'exemple suivant :

Exemple 2.16 *Soit A l'automate de la Figure 2.11 et soit A_1, A_2, A_3 et A_4 les automates de la Figure 2.12. On peut vérifier que $L(A) = L(A_1) \cup L(A_2) \cup L(A_3) \cup L(A_4)$.*

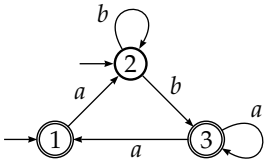


FIGURE 2.11 – *L'Automate A.*

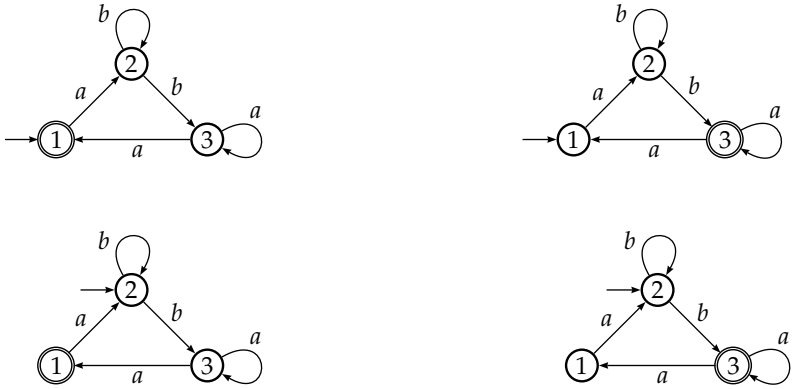


FIGURE 2.12 – *Décomposition de A en A_1, A_2, A_3 et A_4.*

Soit A un automate et soit p et q deux états de A. L'ensemble des mots étiquetant un chemin allant de p à q est noté $L_{p,q}(A)$, et nous notons $E_{p,q}(A)$ une expression dénotant $L_{p,q}(A)$. Cette définition mène alors aux deux lemmes suivants :

Lemme 2.7 *Soit $A = (\Sigma, Q, I, F, \delta)$ un NFA et soit p et q deux états de Q. Considérons le NFA $A' = (\Sigma, Q, \{p\}, \{q\}, \delta)$. Alors :*

$$L_{p,q}(A) = L(A').$$

Démonstration. Par définition,
$$L_{p,q}(A) = \{w \in \Sigma^* \mid q \in \delta(p,w)\} = L(A').$$
□

Lemme 2.8 *Soit $A = (\Sigma, Q, I, F, \delta)$ un NFA. Alors :*
$$L(A) = \bigcup_{(i,f)\in I\times F} L_{i,f}(A).$$

Démonstration. Par définition,
$$L(A) = \{w \in \Sigma^* \mid \delta(I,w) \cap F \neq \emptyset\} = \bigcup_{(i,f)\in I\times F} L_{i,f}(A).$$
□

Ainsi, pour calculer le langage reconnu par un automate, il suffit de calculer l'union des langages définie par le Lemme 2.8. Afin d'obtenir un calcul récursif de ces langages, nous considérons que les états de l'automate A sont ordonnés. Soit $Q = \{q_1, \ldots, q_n\}$ l'ensemble des états avec pour tout q_k, $q_{k'}$ de Q, $q_k \leq q_{k'} \Leftrightarrow k \leq k'$. À partir de cet ordre, deux suites particulières sont définies. La suite $\mathcal{L}(A)$ contient les langages $L^k_{p,q}$ avec $p, q \in Q$, $k \in [\![1, n]\!]$ tels que $L^k_{p,q}$ est l'ensemble des mots étiquetant un chemin allant de p vers q en ne passant que par des états intermédiaires inférieurs à k. La suite $\mathcal{E}(A)$ contient les expressions rationnelles $E^k_{p,q}$ avec $p, q \in Q$, $k \in [\![1, n]\!]$ telles que $E^k_{p,q}$ dénote le langage $L^k_{p,q}$.

Ainsi, pour calculer le langage $L_{p,q}(A)$, il suffit de calculer le langage $L^n_{p,q}$, où n est le nombre d'états de A. Le principe de récurrence est défini comme suit :

Lemme 2.9 *Soit $A = (\Sigma, Q, I, F, \delta)$ un NFA tel que l'ensemble Q soit ordonné. Les suites $\mathcal{L}(A) = (L^k_{p,q})_{p,q\in Q, k\in[\![0,n]\!]}$ et $\mathcal{E}(A) = (E^k_{p,q})_{p,q\in Q, k\in[\![0,n]\!]}$ sont définies récursivement comme suit :*
$$L^k_{p,q} = \begin{cases} \{a \mid (p,a,q) \in \delta\} \cup \{\varepsilon\} & \text{si } k = 0 \text{ et si } p = q \\ \{a \mid (p,a,q) \in \delta\} & \text{si } k = 0 \text{ et si } p \neq q \\ L^{k-1}_{p,q} \cup L^{k-1}_{p,q_k} \cdot (L^{k-1}_{q_k,q_k})^* \cdot L^{k-1}_{q_k,q} & \text{sinon.} \end{cases}$$
$$E^k_{p,q} = \begin{cases} \varepsilon + \bigcup_{a|(p,a,q)\in\delta} a & \text{si } k = 0 \text{ et si } p = q \\ \bigcup_{a|(p,a,q)\in\delta} a & \text{si } k = 0 \text{ et si } p \neq q \\ E^{k-1}_{p,q} + E^{k-1}_{p,q_k} \cdot (E^{k-1}_{q_k,q_k})^* \cdot E^{k-1}_{q_k,q} & \text{sinon.} \end{cases}$$

Démonstration. Pour tout k de $[\![0, n]\!]$, pour tout p, q de Q, les langages $L(E^k_{p,q})$ et $L^k_{p,q}$ sont égaux par construction. On peut résumer le passage de $L^{k-1}_{p,q}$ à $L^k_{p,q}$ dans cet algorithme par le schéma de la Figure 2.13. Et ainsi, selon la définition de $L^k_{p,q}$, les formules sont correctes. □

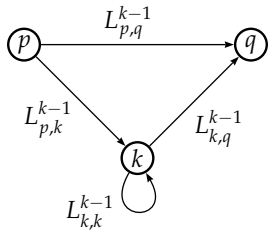


FIGURE 2.13 – *Principe de Récurrence de l'Algorithme MNY.*

Exemple 2.17 *Reprenons l'automate A de la Figure 2.11. La suite $\mathcal{E}(A)$ est la suivante :*

$$\begin{array}{lll} E_{1,1}^0 = \varepsilon & E_{2,1}^0 = \emptyset & E_{3,1}^0 = a \\ E_{1,2}^0 = a & E_{2,2}^0 = b + \varepsilon & E_{3,2}^0 = \emptyset \\ E_{1,3}^0 = \emptyset & E_{2,3}^0 = b & E_{3,3}^0 = a + \varepsilon \end{array}$$

$$\begin{array}{ll} E_{1,1}^1 = E_{1,1}^0 + E_{1,1}^0 \cdot (E_{1,1}^0)^* \cdot E_{1,1}^0 & E_{2,1}^1 = E_{2,1}^0 + E_{2,1}^0 \cdot (E_{1,1}^0)^* \cdot E_{1,1}^0 \\ \quad = \varepsilon & \quad = \emptyset \\ E_{1,2}^1 = E_{1,2}^0 + E_{1,1}^0 \cdot (E_{1,1}^0)^* \cdot E_{1,2}^0 & E_{2,2}^1 = E_{2,2}^0 + E_{2,1}^0 \cdot (E_{1,1}^0)^* \cdot E_{1,2}^0 \\ \quad = a & \quad = b + \varepsilon \\ E_{1,3}^1 = E_{1,3}^0 + E_{1,1}^0 \cdot (E_{1,1}^0)^* \cdot E_{1,3}^0 & E_{2,3}^1 = E_{2,3}^0 + E_{2,1}^0 \cdot (E_{1,1}^0)^* \cdot E_{1,3}^0 \\ \quad = \emptyset & \quad = b \end{array}$$

$$\begin{array}{l} E_{3,1}^1 = E_{3,1}^0 + E_{3,1}^0 \cdot (E_{1,1}^0)^* \cdot E_{1,1}^0 \\ \quad = a \\ E_{3,2}^1 = E_{3,2}^0 + E_{3,1}^0 \cdot (E_{1,1}^0)^* \cdot E_{1,2}^0 \\ \quad = aa = a^2 \\ E_{3,3}^1 = E_{3,3}^0 + E_{3,1}^0 \cdot (E_{1,1}^0)^* \cdot E_{1,3}^0 \\ \quad = a + \varepsilon \end{array}$$

$$\begin{array}{ll} E_{1,1}^2 = E_{1,1}^1 + E_{1,2}^1 \cdot (E_{2,2}^1)^* \cdot E_{2,1}^1 & E_{2,1}^2 = E_{2,1}^1 + E_{2,2}^1 \cdot (E_{2,2}^1)^* \cdot E_{2,1}^1 \\ \quad = \varepsilon & \quad = \emptyset \\ E_{1,2}^2 = E_{1,2}^1 + E_{1,2}^1 \cdot (E_{2,2}^1)^* \cdot E_{2,2}^1 & E_{2,2}^2 = E_{2,2}^1 + E_{2,2}^1 \cdot (E_{2,2}^1)^* \cdot E_{2,2}^1 \\ \quad = a + a(b+\varepsilon)^*(b+\varepsilon) & \quad = b + \varepsilon + (b+\varepsilon)(b+\varepsilon)^*(b+\varepsilon) \\ \quad = ab^* & \quad = b^* \\ E_{1,3}^2 = E_{1,3}^1 + E_{1,2}^1 \cdot (E_{2,2}^1)^* \cdot E_{2,3}^1 & E_{2,3}^2 = E_{2,3}^1 + E_{2,2}^1 \cdot (E_{2,2}^1)^* \cdot E_{2,3}^1 \\ \quad = a(b+\varepsilon)^* b & \quad = b + (b+\varepsilon)(b+\varepsilon)^* b \\ \quad = ab^+ & \quad = b^+ \end{array}$$

$$\begin{array}{l} E_{3,1}^2 = E_{3,1}^1 + E_{3,2}^1 \cdot (E_{2,2}^1)^* \cdot E_{2,1}^1 \\ \quad = a \\ E_{3,2}^2 = E_{3,2}^1 + E_{3,2}^1 \cdot (E_{2,2}^1)^* \cdot E_{2,2}^1 \\ \quad = a^2 + a^2(b+\varepsilon)^*(b+\varepsilon) \\ \quad = a^2 b^* \\ E_{3,3}^2 = E_{3,3}^1 + E_{3,2}^1 \cdot (b+\varepsilon)^* \cdot E_{2,3}^1 \\ \quad = a + \varepsilon + a^2(b+\varepsilon)^* b \\ \quad = \varepsilon + a(ab^+ + \varepsilon) \end{array}$$

$$
\begin{aligned}
E^3_{1,1} &= E^2_{1,1} + E^2_{1,3} \cdot (E^2_{3,3})^* \cdot E^2_{3,1} \\
&= \varepsilon + ab^+(\varepsilon + a(ab^+ + \varepsilon))^* a \\
E^3_{1,2} &= E^2_{1,2} + E^2_{1,3} \cdot (E^2_{3,3})^* \cdot E^2_{3,2} \\
&= ab^* + ab^+(\varepsilon + a(ab^+ + \varepsilon))^* a^2 b^* \\
E^3_{1,3} &= E^2_{1,3} + E^2_{1,3} \cdot (E^2_{3,3})^* \cdot E^2_{3,3} \\
&= ab^+ + ab^+(\varepsilon + a(ab^+ + \varepsilon))^* (\varepsilon + a(ab^+ + \varepsilon)) \\
E^3_{2,1} &= E^2_{2,1} + E^2_{2,3} \cdot (E^2_{3,3})^* \cdot E^2_{3,1} \\
&= b^+(\varepsilon + a(ab^+ + \varepsilon))^* a \\
E^3_{2,2} &= E^2_{2,2} + E^2_{2,3} \cdot (E^2_{3,3})^* \cdot E^2_{3,2} \\
&= b^* + b^+(\varepsilon + a(ab^+ + \varepsilon))^* a^2 b^* \\
E^3_{2,3} &= E^2_{2,3} + E^2_{2,3} \cdot (E^2_{3,3})^* \cdot E^2_{3,3} \\
&= b^+ + b^+(\varepsilon + a(ab^+ + \varepsilon))^* (\varepsilon + a(ab^+ + \varepsilon)) \\
E^3_{3,1} &= E^2_{3,1} + E^2_{3,3} \cdot (E^2_{3,3})^* \cdot E^2_{3,1} \\
&= a + \varepsilon + a(ab^+ + \varepsilon)(\varepsilon + a(ab^+ + \varepsilon))^* a \\
E^3_{3,2} &= E^2_{3,2} + E^2_{3,3} \cdot (E^2_{3,3})^* \cdot E^2_{3,2} \\
&= a^2 b^* + \varepsilon + a(ab^+ + \varepsilon)(\varepsilon + a(ab^+ + \varepsilon))^* a^2 b^* \\
E^3_{3,3} &= E^2_{3,3} + E^2_{3,3} \cdot (E^2_{3,3})^* \cdot E^2_{3,3} \\
&= \varepsilon + a(ab^+ + \varepsilon) + (\varepsilon + a(ab^+ + \varepsilon))(\varepsilon + a(ab^+ + \varepsilon))^* (\varepsilon + a(ab^+ + \varepsilon))
\end{aligned}
$$

Pour calculer une expression rationnelle dénotant le langage de l'automate, il suffit de calculer l'union des expressions $E^n_{p,q}$ pour tous les couples (p, q) de $I \times F$. En effet :

$$L(A) = \bigcup_{(i,f) \in I \times F} L_{i,f}(A) = \bigcup_{(i,f) \in I \times F} L^n_{i,f}.$$

Ainsi, l'automate de la Figure 2.11 reconnaît le langage dénoté par l'expression E définie par :

$$
\begin{aligned}
E &= E_{1,1} + E_{1,3} + E_{2,1} + E_{2,2} \\
&= E^3_{1,1} + E^3_{1,3} + E^3_{2,1} + E^3_{2,2}
\end{aligned}
$$

Avec :

$$
\begin{aligned}
E^3_{1,1} &= \varepsilon + ab^+(\varepsilon + a(ab^+ + \varepsilon))^* a, \\
E^3_{1,3} &= ab^+ + ab^+(\varepsilon + a(ab^+ + \varepsilon))^* (\varepsilon + a(ab^+ + \varepsilon)), \\
E^3_{2,1} &= b^+(\varepsilon + a(ab^+ + \varepsilon))^* a, \\
E^3_{2,2} &= b^* + b^+(\varepsilon + a(ab^+ + \varepsilon))^* a^2 b^*.
\end{aligned}
$$

2.2.3 Méthode BMC

La troisième méthode de conversion est une méthode par élimination d'états et de transitions, due à Brzozowski et McCluskey [10]. Cette méthode utilise les *automates généralisés* :

Définition 2.12 *Un* automate généralisé *A sur un alphabet Σ est un automate où les transitions sont étiquetées par des langages (les transitions appartiennent à $Q \times 2^{\Sigma^*} \times Q$ où Q est l'ensemble des états de l'automate).*

Définition 2.13 *Soit Σ un alphabet et soit $A = (2^{\Sigma^*}, Q, I, F, \delta)$ un automate généralisé. Le langage de A est défini comme suit :*

$$L(A) = \{w \in \Sigma^* \mid \exists L \in 2^{\Sigma^*}, w \in L, \delta(I, L) \cap F \neq \emptyset\}.$$

Un automate généralisé avec un nombre d'états fini où les transitions sont étiquetées par des langages rationnels est appelé *automate généralisé fini rationnel*(**AGFR**).

Proposition 2.4 *Soit A un AGFR sur un alphabet Σ. Alors on a :*
$$L(A) \in \mathrm{Rat}(\Sigma^*).$$

Démonstration. En appliquant sur A l'algorithme MNY ou l'algorithme résolution d'équations des sous-sections 2.2.1 et 2.2.2, on obtient une expression rationnelle E sur un alphabet fini inclus dans 2^{Σ^*}. Remarquons que pour tout mot w de Σ^*, $w \in L(A)$ si et seulement si $\exists L \in L(E) \mid w \in L$. Ainsi, $L(A) = \bigcup_{L \in L(E)} L$ (remarquons que cela est différent de $\bigcup_{L \in L(E)} \{L\}$ qui est équivalent à $L(E)$). Puisque chacune des lettres de E est un langage rationnel L, il existe une expression rationnelle E_L dénotant L selon le Lemme 1.1. Ainsi, en substituant L par E_L dans E, pour toute lettre L, on obtient alors une expression rationnelle E' sur l'alphabet Σ. De plus, E' est telle que pour tout mot $w \in \Sigma^*$, $w \in L(E')$ si et seulement si $\exists L \in L(E) \mid w \in L$. Ainsi, $L(A) = L(E')$, et par conséquent $L(A)$ est rationnel. □

À partir de ces notions, on peut calculer pour un NFA A un AGFR A' tel que $L(A') = L(A)$ et tel que A' ne possède que deux états, dont un est initial et l'autre est terminal. Tout NFA A admet un AGFR équivalent avec un unique état initial et un unique état terminal, tous deux distincts :

Définition 2.14 *Soit $A = (\Sigma, Q, I, F, \delta)$ un NFA et $A' = (\Sigma', Q', I', F', \delta')$ l'AGFR défini par :*
- $\Sigma' = \{\varepsilon\} \cup \bigcup_{a \in \Sigma} \{a\}$,
- $Q' = Q \cup \{i, f\}$ *avec* $i, f \notin Q$,
- $I' = \{i\}$,
- $F' = \{f\}$,
- $\delta' = \{(q, \{a\}, q') \in Q' \times \Sigma' \times Q' \mid (q, a, q) \in \delta\}$
  $\cup \bigcup_{i' \in I} (i, \varepsilon, i') \cup \bigcup_{f' \in F} (f', \varepsilon, f)$.

L'AGFR A' est appelé l'AGFR équivalent à A.

Lemme 2.10 *Soit A un NFA et soit A' l'AGFR équivalent à A. Alors :*
$$L(A) = L(A').$$

Démonstration. Par construction, pour tout mot w de Σ^*, w est dans $L(A)$ si et seulement si $\varepsilon \cdot w \cdot \varepsilon$ est dans $L(A')$. Par conséquent, $L(A) = L(A')$. □

Deux transformations différentes sont alors définies, permettant d'éliminer des transitions ou des états d'un AGFR afin d'obtenir un AGFR réduit équivalent. Soit $A' = (\Sigma', Q', I', F', \delta')$ un AGFR. La premiére règle, appelée par la suite R_1, permet de réduire le nombre de transitions de δ' en les regroupant. Si entre deux états p, q de Q, il existe plusieurs transitions, on peut alors les regrouper, comme le montre la Figure 2.14. La

seconde règle, R_2, permet d'éliminer des états en recombinant les transitions, comme le montre la Figure 2.15.

FIGURE 2.14 – *Règle R_1 : Réduction des Transitions.*

FIGURE 2.15 – *Règle R_2 : Élimination d'État.*

Lemme 2.11 *Soit A un AGFR sur un alphabet Σ. Soit A' (resp. A'') un AGFR obtenu par R_1-réduction (resp. par R_2-réduction) de A. Alors :*

$$L(A) = L(A') = L(A'').$$

Démonstration. Soit c un chemin réussi de A d'étiquette L. Par construction, on peut faire correspondre à c un chemin réussi c' de A' (resp. A'') d'étiquette L' telle que $L \subset L'$.

Soit w un mot de $L(A')$ (resp. $L(A'')$). Par définition, w appartient à un langage qui est l'étiquette d'un chemin de A' (resp. A''). Le mot w peut être décomposé selon les étiquettes de A' (resp. A''), et par construction cette décomposition permet de trouver un chemin dans A tel que l'étiquette de ce chemin contienne w.

Par conséquent, $L(A) = L(A') = L(A'')$. □

Soit A un NFA et soit A' l'AGFR équivalent à A ($L(A) = L(A')$ selon le Lemme 2.10). Puisque l'automate A' ne possède qu'un unique état initial non entrant et un unique état final non sortant, tous deux distincts, et que l'application des règles R_1 et R_2 ne modifie pas cette propriété, on peut toujours appliquer au moins une des deux règles R_1 et R_2 si l'automate a au moins trois états. De plus, selon le Lemme 2.11, ces réductions conservent le langage. Par conséquent, on obtient par application de ces règles un automate à deux états, possédant une unique transition les liant, et tel que cette transition soit étiquetée par une expression rationnelle dénotant le langage.

Exemple 2.18 *Considérons l'automate A de la Figure 2.11. Soit A' l'AGFR équivalent à A, représenté Figure 2.16. Par R_2 sur 2, on obtient l'automate représenté Figure 2.17.*

Par R_2 sur 3, on obtient l'automate représenté Figure 2.18. Par R_1 sur $(i,1)$ et sur $(1,f)$, on obtient l'automate représenté Figure 2.19. Par R_2 sur 1, on obtient l'automate représenté Figure 2.20. Par R_1 sur (i,f), on obtient l'automate représenté Figure 2.21. Le langage $L(A)$ est dénoté par l'expression pondérant la transition (i,f), à savoir $(b^*ba^*a+\varepsilon)(ab^*ba^*a)^*(ab^*ba^*a+\varepsilon)+b^*ba^*$.

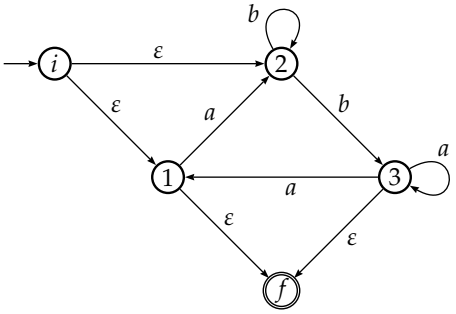


FIGURE 2.16 – *L'AGFR Équivalent à A.*

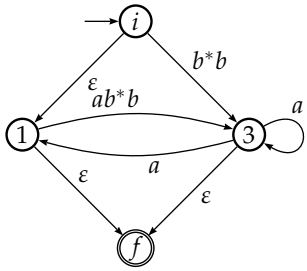


FIGURE 2.17 – *Élimination de* 2.

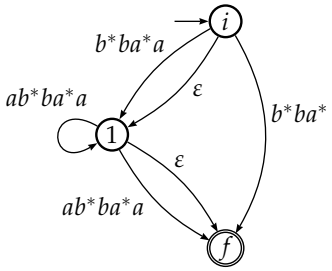


FIGURE 2.18 – *Élimination de* 3.

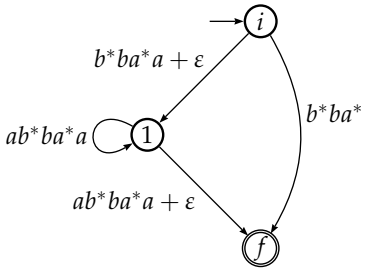


FIGURE 2.19 – *Groupement des Transitions sur $(i,1)$ et $(1,f)$.*

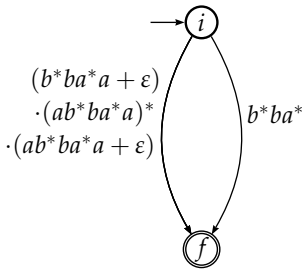


FIGURE 2.20 – *Élimination de* 1.

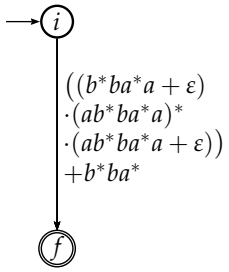


FIGURE 2.21 – *Groupement des Transitions sur* (i, f).

2.2.4 Méthode CZ

La dernière méthode présentée est celle de Caron et Ziadi [13] basée sur la caractérisation des automates construits selon les fonctions de Glushkov, décrites dans la Sous-Section 2.1.1 : si un automate A est un automate vérifiant la caractérisation du Théorème 2.7, cet automate est l'automate de Glushkov d'une expression rationnelle E. De plus, une conversion est possible afin d'obtenir une expression rationnelle E' telle que l'automate des positions de E' soit isomorphe à A.

Tout d'abord, certaines propriétés triviales sont vérifiées par un automate de Glushkov. Selon la méthode de Glushkov présentée dans la Sous-Section 2.1.1, l'automate des positions d'une expression rationnelle E est homogène et standard. Par application de l'algorithme 1.7, on obtient à partir d'un NFA A un NFA homogène A' équivalent. Par application de l'algorithme 1.6, on obtient à partir de A' un NFA A'' standard équivalent. Remarquons que l'ajout du nouvel état initial non entrant conserve l'homogénéité, et ainsi A'' est également homogène. Par conséquent, nous supposerons dans la suite de cette section que les automates utilisés sont standards et homogènes.

Nous utilisons par la suite des *graphes orientés*, que l'on peut voir comme des automates où les transitions ne sont pas étiquetées, et où les états ne sont ni finaux ni initiaux. Un graphe orienté G est un couple (X, U) avec X un *ensemble de sommets* et U un *ensemble d'arcs* tel que U est inclus dans $X \times X$. La structure de transitions de l'automate est étudiée en travaillant sur le graphe sous-jacent de celui-ci :

Définition 2.15 *Soit* $A = (\Sigma, Q, I, F, \delta)$ *un NFA et soit* $G = (X, U)$ *le graphe définit par :*
- $X = Q$,
- $U = \{(p, q) \in X \times X \mid \exists a \in \Sigma, (p, a, q) \in \delta\}$.

Le graphe G est appelé graphe sous-jacent *de A.*

Soit A un NFA sur un alphabet Σ. Puisque le graphe sous-jacent de A est utilisé, le fait qu'un état soit terminal peut être pris en compte par l'ajout de transitions particulières vers un nouvel état qui devient alors

l'unique état terminal de l'automate. Ces transitions sont étiquetées par un symbole n'appartenant pas à l'alphabet de l'automate. Soit # ce symbole. Soit A' ce nouvel automate sur l'alphabet $\Sigma \cup \{\#\}$. Ainsi, pour tout mot de Σ^*, le mot w est dans $L(A)$ si et seulement si le mot $w \cdot \#$ est dans $L(A')$. Puisque cette transformation conserve l'homogénéité, nous considérons alors que tout automate possède un unique état terminal. Cette transformation permet de lier la structure de transitions d'un automate de Glushkov avec la notion de *hamac* :

Définition 2.16 *Soit $G = (X, U)$ un graphe. Le graphe G est un* hamac *si et seulement si une des deux propriétés suivantes est vérifiée :*

(1) *$\#X = 1$ et $U = \emptyset$,*

(2) *$\#X \geq 2$ et il existe i, f deux sommets distincts de X tels que pour tout x de X, il existe un chemin de i vers f dans G passant par x, et il n'existe pas de chemin dans G ni de x vers i, ni de f vers x.*

Proposition 2.5 (Caron et Ziadi [13]) *Soit E une expression rationnelle, et soit A son automate des positions. Alors :*

le graphe sous-jacent de A est un hamac.

Une autre propriété des automates de Glushkov est due à la présence possible d'étoiles dans l'expression rationnelle. Nous nous intéressons alors à des composantes fortement connexes particulières, appelées *orbites* et définies comme suit :

Définition 2.17 *Soit $G = (X, U)$ un graphe et soit $O \subset X$ un sous-ensemble des sommets de G. L'ensemble O est une* orbite *si et seulement si pour tout couple de sommets (x, x') de $O \times O$, il existe un chemin non-vide de x vers x'.*

Si pour tout sommet x de O, pour tout sommet x' de $X \setminus O$, il n'existe pas de chemin de x vers x' ou de x' vers x, l'orbite O est une orbite maximale.

Une orbite est une composante fortement connexe, mais l'inverse n'est pas vrai, puisqu'un sommet isolé sans boucle est une composante fortement connexe, mais que ce sommet n'est pas une orbite.

Soit $G = (X, U)$ un graphe, et soit x un sommet de X. L'ensemble des successeurs (resp. prédécesseurs) directs de x dans G est noté $Q^+(x)$ (resp. $Q^-(x)$). Soit O une orbite de G. L'ensemble $Q^+(x) \cap (X \setminus O)$ (resp. l'ensemble $Q^-(x) \cap (X \setminus O)$) représentant les successeurs (resp. prédécesseurs) directs d'un état x de l'orbite O qui ne sont pas dans O est noté $O^+(x)$ (resp. $O^-(x)$). La *sortie* (resp. l'*entrée*) de l'orbite O est l'ensemble de sommets $\mathrm{Out}(O) = \{x \in O \mid O^+(x) \neq \emptyset\}$ (resp. l'ensemble de sommets $\mathrm{In}(O) = \{x \in O \mid O^-(x) \neq \emptyset\}$). L'orbite O est *stable* si $\mathrm{Out}(O) \times \mathrm{In}(O) \subset U$. L'orbite O est *transversale* si pour tout y, y' de $\mathrm{Out}(O)$ (resp. pour tout y, y' de $\mathrm{In}(O)$), $O^+(y) = O^+(y')$ (resp. $O^-(y) = O^-(y')$). Soit O une orbite maximale. L'orbite O est *fortement stable* si elle est stable

et qu'après avoir éliminé tout arc de $\mathrm{Out}(O) \times \mathrm{In}(O)$, toute sous-orbite maximale est fortement stable. L'orbite O est *fortement transversale* si elle est transversale et qu'après avoir éliminé tout arc de $\mathrm{Out}(O) \times \mathrm{In}(O)$, toute sous-orbite maximale est fortement transversale.

Lemme 2.12 (Caron et Ziadi [13]) *Soit E une expression rationnelle, et soit A son automate des positions. Alors :*
toute orbite maximale du graphe sous-jacent de A est stable et transversale.

Une fois le cas des orbites réglé, il faut traiter la structure de transitions du graphe sans-orbite, défini comme suit :

Définition 2.18 *Soit $G = (X, U)$ un graphe tel que toute orbite de G soit stable. Le* graphe sans orbite *de G, noté $SO(G)$, est le graphe obtenu par élimination récursive pour chacune des orbites maximales O de G des arcs* $\mathrm{Out}(O) \times \mathrm{In}(O)$.

Soit A un automate tel que le graphe sous-jacent $G = (X, U)$ de A soit sans orbite. L'automate A est *réductible* si son graphe sous-jacent G peut être réduit en un unique sommet par une suite d'applications des règles R_1, R_2 et R_3 représentées respectivement Figure 2.22, Figure 2.23 et Figure 2.24 et définies comme suit :

- R_1 : Si x et y sont deux sommets tels que $Q^-(y) = \{x\}$ et $Q^+(x) = \{y\}$, on élimine y et on remplace $Q^+(x)$ par $Q^+(y)$.

- R_2 : Si x et y sont deux sommets tels que $Q^-(x) = Q^-(y)$ et $Q^+(x) = Q^+(y)$, on élimine y ainsi que tout arc connecté à y.

- R_3 : Si x est un sommet tel que pour tout y de $Q^-(x)$, $Q^+(x) \subset Q^+(y)$, on élimine tout arc (q^-, q^+) de $Q^-(x) \times Q^+(x)$ s'il n'existe pas de sommet z dans $X \setminus \{x\}$ tel que les conditions suivantes sont vérifiées :
  - il n'existe pas de chemin de x à z ni de z à x,
  - $q^- \in Q^-(z)$ et $q^+ \in Q^+(z)$,
  - $\#Q^-(z) \times \#Q^+(z) \neq 1$.

FIGURE 2.22 – *Règle R_1*

FIGURE 2.23 – *Règle R_2*

FIGURE 2.24 – *Règle R_3*

À partir de ces propriétés, les automates de Glushkov sont caractérisés comme suit :

Théorème 2.7 (Caron et Ziadi [13]) *Soit A un NFA standard et homogène et soit G son graphe sous-jacent. L'automate A est un automate de Glushkov si et seulement si les trois propositions suivantes sont vérifiées :*

(1) *G est un hamac,*

(2) *Toute orbite maximale de G est fortement stable et fortement transversale,*

(3) *Le graphe sans orbite de G est réductible.*

Cette caractérisation définit un algorithme transformant tout automate de Glushkov A en une expression rationnelle E telle que A soit isomorphe à l'automate des positions de E. Illustrons cet algorithme par l'exemple suivant :

Exemple 2.19 *Considérons l'automate A présenté Figure 2.25. Son graphe sous-jacent G est représenté Figure 2.26. Remarquons que chacune des orbites maximales de G est fortement stable et fortement transversale. Son graphe sans-orbite $SO(G)$ est représenté Figure 2.27. Par R_2-réduction sur les sommets 2 et 3, on obtient le graphe de la Figure 2.28. Par R_3-réduction sur le sommet 1, on obtient le graphe de la Figure 2.29. Par R_1-réduction on réduit le graphe à un unique état représensenté par la la Figure 2.30, et l'expression du graphe sans orbite est alors $E' = (a + \varepsilon)(b + c)\#$. En considérant les orbites, on obtient alors l'expression $E = (a + \varepsilon)^*(b + c)\#$ dénotant $L(A)$.*

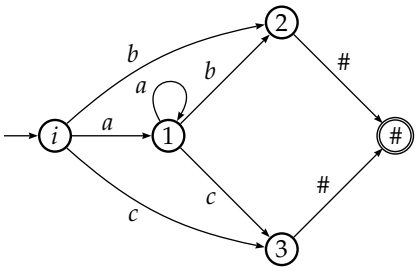


FIGURE 2.25 – *L'Automate A.*

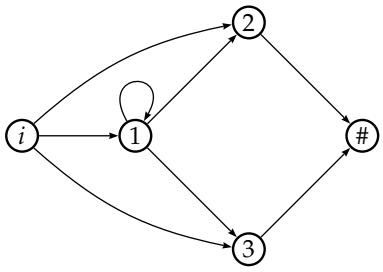


FIGURE 2.26 – *Le Graphe Sous-Jacent G de A.*

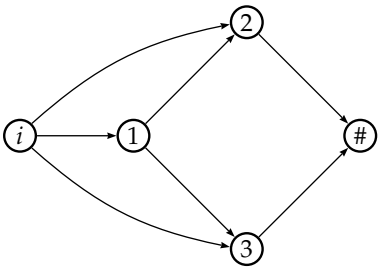


FIGURE 2.27 – *Le Graphe Sans-Orbite* $SO(G)$.

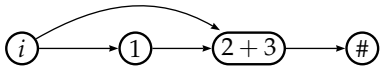


FIGURE 2.28 – R_2*-Réduction sur* 2 *et* 3.

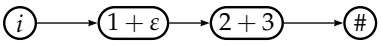


FIGURE 2.29 – R_3*-Réduction sur* 1.

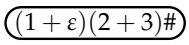


FIGURE 2.30 – R_1*-Réduction.*

2.3 Conclusion

Il existe de nombreuses méthodes de conversion entre NFA et expressions rationnelles simples. Le lien entre ces deux structures est un domaine particulièrement étudié. Différents auteurs ont proposé des extensions de la conversion d'une expression en un automate aux expressions rationnelles à multiplicités (construction de Glushkov par Caron et Flouret dans [12], automate des dérivées partielles par Lombardy et Sakarovitch dans [33], amélioration de la complexité de la construction de Glushkov par Champarnaud, Laugerotte, Ouardi et Ziadi dans [14], c-continuations par Champarnaud, Ouardi et Ziadi dans [15]).

Nous montrerons dans les chapitres suivants l'extension des méthodes de conversion vers et depuis une expression rationnelle simple à de nouveaux types d'expressions rationnelles, les multi-barres, les multi-tildes, et les multi-tildes-barres.

Chapitre 3

Les Multi-Barres et les Multi-Tildes

Un ordinateur fait au bas mot un million d'opérations à la seconde,
mais il a que ça à penser, aussi.
Jean-Marie Gourio

Tout le succès d'une opération réside dans sa préparation.
Sun Tzu

Sommaire

Nous définissons dans ce chapitre deux nouvelles familles d'opérateurs rationnels et compatibles avec la linéarisation. Ces opérateurs sont basés sur des opérations élémentaires, et nous permettent de réduire la largeur des expressions rationnelles. La Section 3.1 est une introduction présentant les notions que nous allons mettre en place. Les notations mathématiques particulières utilisées dans ce chapitre sont présentées dans la Section 3.2. Les deux nouvelles familles d'opérateurs sont présentées dans la Section 3.3. Nous montrons dans la Section 3.4 et dans la Section 3.5 que ces opérateurs sont rationnels et compatibles avec la linéarisation. Enfin, nous définissons un nouveau type d'expressions rationnelles dans la Section 3.6.

3.1 Introduction

Soit E une expression rationnelle simple. Ziadi a étudié dans [46] la complexité de la construction d'une expression rationnelle F telle que $L(F) = L(E) \setminus \{\varepsilon\}$. Il a démontré que lors ce cette construction, la largeur alphabétique de l'expression F est majorée par $\frac{|E|+1}{2}\log(|E|+1) + \frac{|E|-1}{2}$. Ainsi, une opération relativement simple sur un automate A, à savoir la transformation de l'état initial et terminal en état initial et non terminal peut conduire à une augmentation au plus logarithmique de la largeur d'une expression dénotant $L(A)$.

Un nouvel opérateur unaire est introduit, la *barre*, noté $\overline{}$ et défini par $L(\overline{E}) = L(E) \setminus \{\varepsilon\}$. Cet opérateur rationnel est compatible avec la notion de linéarisation utilisée lors de la construction de Glushkov.

Lemme 3.1 *Soit $\mathcal{O} = \{+, \cdot, ^*, \overline{}\}$ un ensemble d'opérateurs. Les opérateurs de $\mathcal{O}$ sont compatibles avec la linéarisation.*

Démonstration. Soit E une expression rationnelle sur les opérateurs de $\mathcal{O}$, et soit $E^{\#}$ l'expression linéarisée de E. Nous étendons l'induction présentée dans la preuve du Lemme 1.9. Supposons que $E = \overline{F}$. Posons $E^{\#} = \overline{F'}$. Par induction, $L(F) = \mathrm{h}_E(L(F'))$. Par définition, on a $L(\overline{F}) = L(F) \setminus \{\varepsilon\}$. Par conséquent :

$$\begin{aligned} L(E) = L(\overline{F}) &= L(F) \setminus \{\varepsilon\} \\ &= \mathrm{h}_E(L(F')) \setminus \{\varepsilon\} \\ &= \mathrm{h}_E(L(F') \setminus \{\varepsilon\}) \\ &= \mathrm{h}_E(L(\overline{F'})) \qquad = \mathrm{h}_E(L(E^{\#})). \end{aligned}$$

□

Lemme 3.2 *Soit E une expression rationnelle et soit x un élément de $\mathrm{Pos}(E)$. Les fonctions de Glushkov de l'expression $\overline{E}$ sont calculées comme suit :*

$$\mathrm{Pos}(\overline{E}) = \mathrm{Pos}(E), \quad \mathrm{Null}(\overline{E}) = \emptyset,$$
$$\mathrm{First}(\overline{E}) = \mathrm{First}(E), \quad \mathrm{Last}(\overline{E}) = \mathrm{Last}(E),$$
$$\mathrm{Follow}(\overline{E}, x) = \mathrm{Follow}(E, x).$$

Démonstration. Soit w dans Σ^+. Selon la définition de l'opérateur barre, on a $w \in L(E) \Leftrightarrow w \in L(\overline{E})$. Les expressions E et $\overline{E}$ admettent donc les mêmes fonctions Pos, First, Last et Follow. Comme $\varepsilon \notin L(\overline{E})$, la fonction Null est correcte. □

Cette compatibilité avec les positions permet de calculer très facilement un automate par la méthode de Glushkov (voir Section 2.1.1) à partir d'une expression rationnelle utilisant les opérateurs $+$, $\cdot$, *, ou $\overline{}$.

L'élimination du mot vide peut être réalisée par l'opérateur de différence ensembliste. Cependant, cet opérateur n'est pas compatible avec la linéarisation.

Exemple 3.1 *Soit $E = (a(a+\varepsilon)) \setminus a$ une expression rationnelle. Soit $E^{\#}$ l'expression linéarisée de E : $E^{\#} = (a_1(a_2+\varepsilon)) \setminus a_3$. Le langage de E est $L(E) = \{aa\}$, alors que le langage de $E^{\#}$ est $L(E^{\#}) = \{a_1, a_1a_2\}$. Par conséquent, $L(E) \neq \mathrm{h}_E(L(E^{\#}))$.*

L'utilisation de la barre permet ainsi d'obtenir un gain au plus logarithmique sur la largeur d'une expression, dû à un gain du même ordre sur chacune des sous-expressions placées sous une barre, tout en conservant la compatibilité avec la linéarisation.

Exemple 3.2 *Soit $E = \overline{(a+\varepsilon)\cdot(b+\varepsilon)} + \overline{(c+\varepsilon)\cdot(d+\varepsilon)}$ une expression de largeur 4 dont le langage est reconnu par l'automate de la Figure 3.1. Le langage dénoté par E, le langage $\{a, b, c, d, ab, cd\}$, est aussi dénoté par l'expression rationnelle simple $E' = a + b + a \cdot b + c + d + c \cdot d$ de largeur 8 ou par l'expression rationnelle simple $E'' = a \cdot (b+\varepsilon) + b + c \cdot (d+\varepsilon) + d$ de largeur 6. Mais aucune expression rationnelle simple de largeur inférieure à 6 ne dénote le langage $L(E)$.*

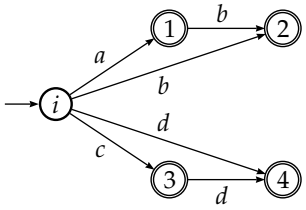


FIGURE 3.1 – *Un Automate Reconnaissant le Langage $L(E) = \{a, b, c, d, ab, cd\}$.*

La barre élimine le mot vide dans une sous-expression d'une expression donnée, mais le fait que cet opérateur soit unaire en restreint considérablement la puissance.

Exemple 3.3 *Considérons les langages $L_1 = \{\varepsilon, a\}$, $L_2 = \{\varepsilon, b\}$, $L_3 = \{\varepsilon, c\}$ et*

$$L = \{w_1w_2w_3 \mid \forall k \in [\![1,3]\!],\ w_k \in L_k \ \wedge\ w_1 \cdot w_2 \neq \varepsilon \ \wedge\ w_2 \cdot w_3 \neq \varepsilon\}.$$

Le langage L est obtenu en calculant la concaténation $L_1 \cdot L_2 \cdot L_3$ et en interdisant le fait que L_1 et L_2 soient substitués par ε en même temps, de même que L_2 et L_3 ; ceci n'est pas possible en utilisant une ou plusieurs barres unaires sur des sous-expressions de $L_1 \cdot L_2 \cdot L_3$. Nous montrerons dans la suite de cette section la signification de la notation :

$$\overline{\overline{(a+\varepsilon)}(b+\varepsilon)}(c+\varepsilon)\ .$$

Les différentes combinaisons de contraintes interdisant que certains facteurs d'un produit de concaténation ne soient substituables par le mot vide sont prises en compte par de nouveaux opérateurs étendant le pou-

voir des barres. Ces opérateurs sont paramétrés par un ensemble de couples qui représentent chacun un facteur non-substituable par le mot vide.

3.2 Quelques Notations Mathématiques

Dans la suite, nous utilisons des ensembles totalement ordonnés. Afin d'alléger les notations, tout ensemble totalement ordonné est appelé *liste*. Pour une liste de couples d'entiers croissants, l'ordre lexicographique usuel des couples est utilisé. Cet ordre est défini comme suit. Soit (i, f) et (i', f') deux couples d'entiers croissants. Le couple (i, f) est inférieur ou égal au couple (i', f') si et seulement si $i < i'$ ou ($i = i'$ et $f \leq f'$). Dans ce cas, nous notons $(i, f) \leq (i', f')$. Pour une liste d'expressions, nous la considérons ordonnée selon l'ordre de sa définition. Ainsi, dans une liste d'expressions $(E_1, E_2, \ldots, E_n)$, on considère l'ordre $E_k \leq E_{k'} \Leftrightarrow k \leq k'$ pour tout (k, k') de $[\![1, n]\!]^2$. Par souci de concision, la liste d'expressions $(E_1, \ldots, E_n)$ est notée $E_{1,n}$. De même, pour une concaténation $E = E_1 \cdots E_n$, l'expression E est notée $E_{1\cdots n}$.

Soit n un entier positif. L'ensemble des listes finies constituées de couples croissants d'entiers compris entre 1 et n est noté $\mathcal{S}_n$. Soit S une liste de $\mathcal{S}_n$. L'ensemble des indices de la liste est $I_S = [\![1, \#S]\!]$. Ainsi, la liste S de $\mathcal{S}_n$ est notée $S = ((i_k, f_k))_{k \in I_S}$, avec pour tout $k \in I_S$, $(i_k, f_k) \in [\![1, n]\!]^2_{\leq}$. On note $\mathrm{Sup}(S)$ (resp. $\mathrm{Inf}(S)$, $\mathrm{Succ}(s)$ et $\mathrm{Pred}(s)$ pour $s \in S$) le plus grand couple de S (resp. le plus petit couple de S, le plus petit couple de S plus grand que s et le plus grand couple de S plus petit que s) s'il existe.

Soit (i, f) un couple de la liste S. Le couple (i, f) représente le sous-intervalle $[\![i, f]\!]$ de $[\![1, n]\!]$. Le couple (i, f) est *chevauché* (ou *chevauchant*) s'il existe un couple (i', f') de S tel que $i' < i < f' < f$ (*chevauché à gauche* ou *chevauchant à droite*) ou $i < i' < f < f'$ (*chevauché à droite* ou *chevauchant à gauche*). On dit alors que le couple (i', f') chevauche (i, f). Le couple (i, f) est *inclus* s'il existe un couple (i', f') distinct de (i, f) tel que $i' \leq i \leq f \leq f'$. On dit alors que le couple (i', f') inclut (i, f). Le couple (i, f) est *surplombant* s'il n'est pas chevauché. La liste S est *libre* si et seulement si pour tout $(i, f), (i', f')$ de S tels que $(i, f) \neq (i', f')$, $[\![i, f]\!] \cap [\![i', f']\!] = \emptyset$. Toute liste de cardinal inférieur à 2 est par conséquent une liste libre. Afin d'expliciter ces notions sur les couples, représentons les graphiquement dans l'exemple suivant :

Exemple 3.4 *Considérons la liste $S_1 = ((1,3),(1,5),(3,5))$ (dont les couples sont représentés Figure 3.2). Les couples $(1,3)$ et $(3,5)$ se chevauchent, et ces deux couples sont inclus dans $(1,5)$ (voir Figure 3.3). Considérons maintenant la liste de couples $S_2 = ((1,1),(2,4),(5,5))$. Par définition, cette liste est libre (voir Figure 3.4). Remarquons que le couple $(1,5)$ est surplombant dans S_1, et que tout couple de S_2 l'est également.*

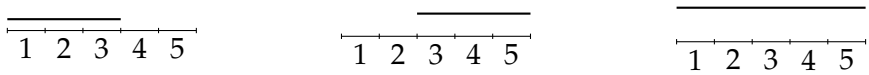


FIGURE 3.2 – *Représentation Graphique des Couples* $(1,3)$, $(3,5)$ *et* $(1,5)$.

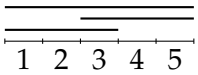


FIGURE 3.3 – *Inclusions et Chevauchements.*

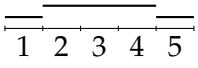


FIGURE 3.4 – *Une Liste Libre.*

3.3 Deux Nouvelles Familles d'Opérateurs...

Dans cette section, nous montrons comment étendre la notion d'élimination du mot vide dans une expression à celle de l'élimination du mot vide dans certains facteurs d'un produit de concaténation, y compris chevauchants. Cette opération est réalisée par les opérateurs de multi-barres (Sous-Section 3.3.1). Cette notion admet une notion duale, à savoir l'ajout du mot vide dans certains facteurs d'un produit de concaténation, réalisée par les opérateurs de multi-tildes (Sous-Section 3.3.2).

3.3.1 Les Multi-Barres

Pour formaliser l'action d'un opérateur composé de plusieurs barres sur une liste d'expressions $E_{1,n}$, l'ensemble des barres est donné par une liste de couples B de $\mathcal{S}_n$; chaque couple (i,f) de B indique une barre sur le facteur $E_{i\cdots f}$, avec $\overline{E_{i,f}}$ pour représentation graphique.

Exemple 3.5 *Le langage L de l'Exemple 3.3 est dénoté par l'expression* $\overline{\overline{(a+\varepsilon)(b+\varepsilon)}(c+\varepsilon)}$. *La barre $(1,2)$ (resp. $(2,3)$) interdit la substitution de $L((a+\varepsilon)(b+\varepsilon))$ (resp. $L((b+\varepsilon)(c+\varepsilon))$) par ε.*

Un mot appartient au langage d'une liste de barres B appliquée sur une liste d'expressions $E_{1,n}$ s'il admet une décomposition $w_1 \cdots w_n$ sur le produit de concaténation $L(E_1) \cdots L(E_n)$ telle que pour toute barre (i,f) de B, le facteur $w_i \cdots w_f$ est différent du mot vide. Cette propriété pour une décomposition d'un mot et pour une barre est appelée *vérification*.

Définition 3.1 *Soit $E_{1,n}$ une liste d'expressions et $w = w_1 \cdots w_n$ un mot de $L(E_{1\cdots n})$ tel que $\forall k \in [\![1,n]\!]$, $w_k \in L(E_k)$. Soit (i,f) un couple de $[\![1,n]\!]^2_{\leq}$. Le mot w* vérifie la barre (i,f) *si et seulement si $w_i \cdots w_f \neq \varepsilon$.*

Des listes de barres différentes appliquées à des listes d'expressions différentes peuvent posséder la même représentation graphique (voir Exemple 3.6). Pour lever cette ambiguïté, nous utilisons un opérateur paramétré appliqué sur une liste d'expressions de cardinal fixé.

Exemple 3.6 *Soit $\mathcal{E}_1 = ((a+\varepsilon),(b+\varepsilon))$ et $\mathcal{E}_2 = ((a+\varepsilon)(b+\varepsilon))$ deux listes d'expressions et soit $B_1 = (1,2)$ et $B_2 = (1,1)$ deux listes de barres. L'application de B_1 sur $\mathcal{E}_1$ et l'application de B_2 sur $\mathcal{E}_2$ possèdent la même représentation graphique, à savoir* $\overline{(a+\varepsilon)(b+\varepsilon)}$.

Définition 3.2 *Soit n un entier positif. Soit $E_{1,n}$ une liste d'expressions sur un alphabet Σ. Soit B une liste de $\mathcal{S}_n$. La multi-barres $E = \text{—}_B(E_{1,n})$ dénote le langage :*

$$L(E) = \left\{ \begin{array}{rl} w \in \Sigma^* \mid \textit{il existe une décomposition de } w \textit{ en } w_1 \cdots w_n \\ \textit{telle que} \quad \forall k \in [\![1,n]\!],\ w_k \in L(E_k) \\ \textit{et telle que} \quad \forall (i,f) \in B,\ (i,f) \textit{ est vérifié par } w_1 \cdots w_n. \end{array} \right\}$$

3.3.2 Les Multi-Tildes

L'opération duale de la barre, l'ajout du mot vide, a peu d'intérêt si elle est considérée comme unaire, puisque pour toute expression rationnelle E, l'ajout du mot vide dans le langage dénoté par E produit l'expression $E+\varepsilon$ de même largeur que E. L'opérateur tilde, noté $\sim$, ajoute le mot vide au langage dénoté par une expression : $L(\widetilde{E}) = L(E) \cup \{\varepsilon\}$. Cet opérateur est compatible avec la notion de linéarisation.

Lemme 3.3 *Soit $\mathcal{O} = \{+, \cdot, ^*, \sim\}$ un ensemble d'opérateurs. Les opérateurs de $\mathcal{O}$ sont compatibles avec la linéarisation.*

Démonstration. Soit E une expression rationnelle sur les opérateurs de $\mathcal{O}$, et soit $E^{\#}$ son expression linéarisée. Nous étendons l'induction présentée dans la preuve du Lemme 1.9. Supposons que $E = \widetilde{F}$. Posons $E^{\#} = \widetilde{\widetilde{F'}}$. Par induction, $L(F) = \mathrm{h}_E(L(F'))$. Par définition, on a $L(\widetilde{F}) = L(F) \cup \{\varepsilon\}$. Par conséquent :

$$\begin{aligned} L(E) = L(\widetilde{F}) &= L(F) \cup \{\varepsilon\} \\ &= \mathrm{h}_E(L(F')) \cup \{\varepsilon\} \\ &= \mathrm{h}_E(L(F') \cup \{\varepsilon\}) \\ &= \mathrm{h}_E(L(\widetilde{F'})) \qquad = \mathrm{h}_E(L(E^{\#})). \end{aligned}$$

□

Lemme 3.4 *Soit E une expression rationnelle et soit x un élément de $\mathrm{Pos}(E)$. Les fonctions de Glushkov de l'expression $\widetilde{E}$ sont calculées comme suit :*

$$\begin{gathered} \mathrm{Pos}(\widetilde{E}) = \mathrm{Pos}(E), \quad \mathrm{Null}(\widetilde{E}) = \{\varepsilon\}, \\ \mathrm{First}(\widetilde{E}) = \mathrm{First}(E), \quad \mathrm{Last}(\widetilde{E}) = \mathrm{Last}(E), \\ \mathrm{Follow}(\widetilde{E}, x) = \mathrm{Follow}(E, x). \end{gathered}$$

Démonstration. Soit w dans Σ^+. Selon la définition de l'opérateur tilde, on a $w \in L(E) \Leftrightarrow w \in L(\widetilde{E})$. Les expressions E et $\widetilde{E}$ admettent donc les

mêmes fonctions Pos, First, Last et Follow. Comme $\varepsilon \in L(\widetilde{E})$, la fonction Null est correcte. □

L'extension de la tilde à la multi-tildes se traite de façon similaire à celle de la barre à la multi-barres. L'ensemble de tildes à appliquer sur une liste d'expressions $E_{1,n}$ est alors donné par une liste T de $\mathcal{S}_n$, où tout couple (i, f) est une tilde appliquée sur $E_{i\cdots f}$, représentée par $\widetilde{E_{i,f}}$.

Exemple 3.7 *Soit $L = \{abcd, ab, cd, ad, \varepsilon\}$ un langage rationnel. Le langage L est dénoté par les expressions E et E′ définies par :*

$$E = abcd + \widetilde{a\varepsilon\varepsilon d} + \widetilde{\varepsilon\varepsilon cd} + \widetilde{ab\varepsilon\varepsilon} + \widetilde{\varepsilon\varepsilon\varepsilon\varepsilon},$$
$$E' = abcd + \widetilde{a\ bc\ d} + \widetilde{ab\ cd} + \widetilde{ab\ cd} + \widetilde{abcd}\ .$$

Nous montrons dans la suite que l'expression $E'' = \widetilde{\widetilde{(a)(b)}(c)(d)}$ dénote le langage L.

L'action de la tilde unaire sur une expression se résume en un choix simple : soit l'expression est inchangée, soit elle est remplacée par le mot vide. L'action d'une liste de tildes est alors une combinaison de ces choix. Pour chaque tilde, soit l'expression qu'elle couvre est dans sa totalité remplacée par le mot vide, soit elle est inchangée. Remarquons alors que si deux tildes se chevauchent, elles ne peuvent pas s'appliquer en même temps, puisque si l'une s'applique, elle modifie un facteur de l'expression couverte par l'autre, qui ne peut plus alors s'appliquer. Par conséquent, seule une combinaison de tildes formant une liste libre peut s'appliquer en même temps. On peut alors calculer l'ensemble de mots générés par l'application d'une telle liste. On définit ainsi la *génération* d'un mot par une liste libre de couples croissants :

Définition 3.3 *Soit n un entier supérieur à 0, soit $E_{1,n}$ une liste d'expressions sur un alphabet Σ et soit T une liste libre de $\mathcal{S}_n$. Soit $\mathcal{W}$ l'ensemble défini par :*

$$\mathcal{W} = \left\{ \begin{array}{l} w \in \Sigma^* \mid \text{il existe une décomposition de } w \text{ en } w_1 \cdots w_n \\ \text{telle que } \ w_k = \varepsilon \text{ si } k \in \bigcup_{(i,f)\in T} [\![i, f]\!],\ w_k \in L(E_k) \text{ sinon.} \end{array} \right\}$$

Soit w un mot de $\mathcal{W}$. On dit alors que w est généré par *T.*

Des listes de tildes différentes appliquées à des listes d'expressions différentes peuvent posséder la même représentation graphique (voir Exemple 3.8). Pour lever cette ambiguïté, nous utilisons un opérateur paramétré appliqué sur une liste d'expressions de cardinal fixé.

Exemple 3.8 *Soit $\mathcal{E}_1 = (a, b)$ et $\mathcal{E}_2 = (ab)$ deux listes d'expressions rationnelles. Considérons les listes de tildes $T_1 = (1, 2)$ et $T_2 = (1, 1)$. Les applications de T_1 sur $\mathcal{E}_1$ et de T_2 sur $\mathcal{E}_2$ possèdent la même représentation graphique, à savoir $\widetilde{ab}$.*

Définition 3.4 *Soit n un entier positif. Soit $E_{1,n}$ une liste d'expressions sur un alphabet Σ. Soit T une liste de $\mathcal{S}_n$. La multi-tildes $E = \sim_T(E_{1,n})$ dénote le langage :*
$$L(E) = \{w \in \Sigma^* \mid w \text{ est généré par une sous-liste libre de } T\}.$$

3.4 ...Compatibles avec la Linéarisation...

La tilde et la barre unaires sont des opérateurs compatibles avec la linéarisation. Nous montrons que cette propriété est également vérifiée par leurs extensions respectives.

Lemme 3.5 *Soit $\mathcal{O}$ l'ensemble des opérateurs défini par :*
$$\mathcal{O} = \{+, \cdot, ^*\} \cup \bigcup_{B \mid \exists n \in \mathbb{N}, B \in \mathcal{S}_n} —_B \cup \bigcup_{T \mid \exists n \in \mathbb{N}, T \in \mathcal{S}_n} \sim_T.$$
Les opérateurs de $\mathcal{O}$ sont compatibles avec la linéarisation.

Démonstration. Soit E une expression sur les opérateurs de $\mathcal{O}$, et soit $E^\#$ son expression linéarisée. Nous étendons l'induction sur la largeur de E et sur son nombre d'opérateurs présentée dans la preuve du Lemme 1.9.

(I) Soit $E = —_B(E_{1,n})$ une multi-barres. Posons $E^\# = —_B(E'_{1,n})$.
(a) Soit $w' = w'_1 \cdots w'_n$ un mot de $L(E^\#)$. Sa décomposition (unique) vérifie chaque barre de B et est telle que pour tout entier k de $[\![1, n]\!]$, w'_k est dans $L(E'_k)$. Soit $w = w_1 \cdots w_n$ le mot obtenu par délinéarisation de w' : pour tout entier k de $[\![1, n]\!]$, $w_k = \mathrm{h}_E(w'_k)$. Puisque w'_k appartient à $L(E'_k)$, par induction, $\mathrm{h}_E(w'_k)$ appartient à $L(E_k)$. De plus, puisque pour toute barre (i, f) de B, $w'_i \cdots w'_f \neq \varepsilon$, $w_i \cdots w_f = \mathrm{h}_E(w'_i) \cdots \mathrm{h}_E(w'_f) \neq \varepsilon$. Et ainsi, selon la Définition 3.2, $w = \mathrm{h}_E(w')$ est dans $L(E)$.
(b) Soit w un mot de $L(E)$. Selon la Définition 3.2, w admet au moins une décomposition $w_1 \cdots w_n$ telle que pour tout entier k de $[\![1, n]\!]$, w_k est dans $L(E_k)$ et pour toute barre (i, f) de B, $w_i \cdots w_f \neq \varepsilon$. De plus, pour tout entier k de $[\![1, n]\!]$, comme w_k est dans $L(E_k)$, par induction il existe nécessairement au moins un mot w'_k appartenant à $L(E'_k)$ tel que $\mathrm{h}_E(w'_k) = w_k$. Ainsi, il existe un mot $w' = w'_1 \cdots w'_n$ tel que pour tout k de $[\![1, n]\!]$, w'_k est dans $L(E'_k)$ et $\mathrm{h}_E(w'_k) = w_k$. D'autre part, puisque pour toute barre (i, f) de B, $w_i \cdots w_f \neq \varepsilon$, $w'_i \cdots w'_f \neq \varepsilon$. Ainsi, $w' \in L(E^\#)$. Et puisque $\mathrm{h}_E(w') = w$, $w \in \mathrm{h}_E(L(E^\#))$.

(II) Soit $E = \sim_T(E_{1,n})$ une multi-tildes. Posons $E^\# = \sim_T(E'_{1,n})$.
(a) Soit w un mot de $L(E)$. Selon la Définition 3.4, il existe une sous-liste libre T' de T générant w. Par conséquent, w se décompose en $w_1 \cdots w_n$ tel que pour tout k de $[\![1, n]\!]$, soit $w_k = \varepsilon$ si $k \in \bigcup_{(i,f) \in T'} [\![i, f]\!]$, soit $w_k \in L(E_k)$. Par induction, pour tout k de $[\![1, n]\!]$, $\mathrm{h}_E(L(E'_k)) = L(E_k)$. Par conséquent, il existe un mot $w' = w'_1 \cdots w'_n$ tel que pour tout k de $[\![1, n]\!]$, soit $w'_k = \varepsilon$ si $k \in \bigcup_{(i,f) \in T'} [\![i, f]\!]$, soit $w'_k \in L(E'_k)$ et $w_k = \mathrm{h}_E(w'_k)$ sinon. Ainsi T' est une sous-liste libre de T générant w'. Selon la Définition 3.4, $w' \in L(E^\#)$. Finalement, $w \in \mathrm{h}_E(L(E^\#))$.

(b) Soit w un mot de $\mathrm{h}_E(L(E)^{\#})$. Alors il existe un mot w' dans $L(E)^{\#}$ tel que $w = \mathrm{h}_E(w')$. Selon la Définition 3.4, il existe T' une sous-liste libre de T générant w'. Par conséquent, w' se décompose en $w'_1 \cdots w'_n$ tel que pour tout k de $[\![1, n]\!]$, soit $w'_k = \varepsilon$ si $k \in \bigcup_{(i,f)\in T'} [\![i, f]\!]$, soit $w'_k \in L(E'_k)$. Par induction, pour tout k de $[\![1, n]\!]$, $\mathrm{h}_E(L(E'_k)) = L(E_k)$. Par conséquent, le mot $w = \mathrm{h}_E(w')$ admet une décomposition $w_1 \cdots w_n$ telle que pour tout k de $[\![1, n]\!]$, soit $w_k = \varepsilon$ si $k \in \bigcup_{(i,f)\in T'} [\![i, f]\!]$, soit $w_k \in L(E_k)$. Selon la Définition 3.4, $w \in L(E)$. □

3.5 ...Et Rationnels...

Dans cette section, nous montrons comment calculer à partir d'opérations rationnelles le langage dénoté par une multi-barres ou une multi-tildes.

3.5.1 Multi-Barres et Expressions Compatibles

Selon la Définition 3.2, un mot appartient au langage d'une expression en multi-barres $E = \text{——}_B(E_{1,n})$ s'il admet une décomposition vérifiant chaque barre de B. Nous décomposons alors l'expression E en une somme d'expressions dans lesquelles chaque expression de la liste $E_{1,n}$ apparaît une et une seule fois, soit sous forme directe, soit sous forme barrée, de telle sorte que tout mot de $L(E')$ vérifie chaque barre de B.

Définition 3.5 *Soit $E = \text{——}_B(E_{1,n})$ une multi-barres. Considérons une expression $E' = E'_{1\cdots n}$ telle que pour tout k de $[\![1, n]\!]$, $E'_k = E_k$ ou $E'_k = \overline{E_k}$. L'expression E' est* compatible *avec l'expression E si et seulement si :*

pour toute barre (i, f) de B, il existe k dans $[\![i, f]\!]$ tel que $E'_k = \overline{E_k}$.

Exemple 3.9 *Considérons l'expression $E = \text{——}_B(E_{1,4})$ avec $B = ((1,2),(2,3),(3,4))$. Les expressions $E_1\,\overline{E_2}\,E_3\,\overline{E_4}$, $E_1\,\overline{E_2}\,\overline{E_3}\,E_4$, $\overline{E_1}\,E_2\,\overline{E_3}\,E_4$ et $\overline{E_1}\,\overline{E_2}\,\overline{E_3}\,\overline{E_4}$ sont compatibles avec E, contrairement aux expressions $E_1E_2E_3E_4$ et $\overline{E_1}\,E_2E_3\,\overline{E_4}$.*

Le langage d'une multi-barres E peut se calculer comme l'union des langages dénotés par les expressions compatibles avec E. Et puisque l'élimination du mot vide est une opération rationnelle, nous en déduisons que le langage dénoté par E est rationnel.

Lemme 3.6 *Soit $E = \text{——}_B(E_{1,n})$ une multi-barres sur un alphabet Σ. Soit w un mot de Σ^*. Alors :*

$$w \in L(E) \Leftrightarrow \text{il existe une expression } E' \text{ compatible avec } E \text{ telle que } w \in L(E').$$

Démonstration. Soit w un mot de $L(E)$. Selon la Définition 3.2, le mot w admet une décomposition $w = w_1 \cdots w_n$ telle que pour tout $k \in [\![1, n]\!]$, $w_k \in L(E_k)$ et telle que pour toute barre (i, f) de B, $w_i \cdots w_f \neq \varepsilon$. En posant $E' = E'_1 \cdots E'_n$ une concaténation d'expressions telle que $E'_k = E_k$ si $w_k = \varepsilon$ et $E'_k = \overline{E_k}$ si $w_k \neq \varepsilon$, E' est une expression telle que $w \in L(E')$. De plus, E' est une expression compatible avec E, puisque par définition, pour toute barre (i, f) de B, il existe un entier k de $[\![i, f]\!]$ tel que $w_k \neq \varepsilon$ et donc tel que $E'_k = \overline{E_k}$. Ainsi, w appartient au langage dénoté par une expression compatible avec E.

Réciproquement, soit $E' = E'_{1\cdots n}$ une expression compatible avec E. Soit $w = w_1 \cdots w_n$ un mot de $L(E')$ tel que pour tout k de $[\![1, n]\!]$, w_k est dans $L(E'_k)$. Puisque E' est compatible avec E, pour chaque barre (i, f) de B, il existe un entier $k \in [\![i, f]\!]$ tel que $E'_k = \overline{E_k}$, et donc tel que $w_k \neq \varepsilon$. Par conséquent, w vérifie chaque barre de B et ainsi appartient à $L(E)$ selon la Définition 3.2. □

Corollaire 3.1 *Soit n un entier positif, $E_{1,n}$ une liste d'expressions rationnelles et $E = \text{——}_B(E_{1,n})$ une multi-barres. Alors :*

le langage dénoté par E est rationnel.

Le langage d'une multi-barres E se calcule à partir des expressions compatibles avec E. En fait, toutes les expressions compatibles ne sont pas nécessaires. Il existe une hiérarchie sur les langages qu'elles dénotent : certains sont inclus dans d'autres. Le calcul des expressions compatibles peut ainsi se restreindre à un ensemble particulier :

Définition 3.6 *Soit $E = \text{——}_B(E_{1,n})$ une multi-barres et soit $E' = E'_{1\cdots n}$ une expression compatible avec E. L'expression compatible E' est* minimale *si pour toute expression $E'' = E''_{1\cdots n}$ différente de E' telle que pour tout k de $[\![1, n]\!]$, $E''_k = E_k$ si $E'_k = E_k$, E'' n'est pas compatible avec E.*

Lemme 3.7 *Soit $E = \text{——}_B(E_{1,n})$ une multi-barres et soit $E' = E'_{1\cdots n}$ une expression compatible avec E. Soit $E'' = E''_{1\cdots n}$ une expression telle que pour tout k de $[\![1, n]\!]$, $E''_k = \overline{E_k}$ si $E'_k = \overline{E_k}$. Les deux propositions suivantes sont vérifiées :*

(1) *E'' est compatible avec E,*

(2) *$L(E'') \subset L(E')$.*

Démonstration. **(1)** Puisque E' est compatible avec E, pour toute barre (i, f) de B, il existe un entier k de $[\![i, f]\!]$ tel que $E'_k = \overline{E_k}$. Ainsi, par définition de E'', pour toute barre (i, f) de B, il existe un entier k de $[\![i, f]\!]$ tel que $E''_k = \overline{E_k}$. L'expression E'' est alors compatible avec E.

(2) Par définition de E'', pour tout entier k de $[\![1, n]\!]$, soit $E'_k = \overline{E_k} = E''_k$, soit $E'_k = E_k$ et $E''_k = E_k$ ou $E''_k = \overline{E_k}$. Par conséquent, pour tout k de $[\![1, n]\!]$,

$L(E''_k) \subset L(E'_k)$. Ainsi, $L(E'') \subset L(E')$. □

Lemme 3.8 *Soit $E = \text{——}_B(E_{1,n})$ une multi-barres et soit $E' = E'_{1\cdots n}$ une expression non-compatible avec E. Soit $E'' = E''_{1\cdots n}$ une expression telle que pour tout k de $[\![1, n]\!]$, $E''_k = E_k$ si $E'_k = E_k$. Les deux propositions suivantes sont vérifiées :*
(1) *E'' n'est pas compatible avec E,*
(2) *$L(E') \subset L(E'')$.*

Démonstration. **(1)** Puisque E' n'est pas compatible avec E, il existe une barre (i, f) de B telle que pour tout k de $[\![i, f]\!]$, $E'_k = E_k$. Ainsi, puisque E'' est défini de telle sorte que pour tout entier k de $[\![1, n]\!]$, $E''_k = E_k$ si $E'_k = E_k$, il existe une barre (i, f) de B telle que pour tout k de $[\![i, f]\!]$, $E''_k = E_k$. Par conséquent E'' n'est pas compatible avec E.

(2) Par définition de E'', pour tout entier k de $[\![1, n]\!]$, soit $E'_k = E_k = E''_k$, soit $E'_k = \overline{E_k}$ et $E''_k = E_k$ ou $E''_k = \overline{E_k}$. Ainsi, pour tout entier k de $[\![1, n]\!]$, $L(E') \subset L(E''_k)$. Ainsi, $L(E') \subset L(E'')$. □

Le langage dénoté par E est ainsi le langage dénoté par la somme des expressions compatibles minimales de E.

Proposition 3.1 *Soit $E = \text{——}_B(E_{1,n})$ une multi-barres sur un alphabet Σ et soit w un mot de Σ^*. Alors :*

$$w \in L(E) \Leftrightarrow \text{il existe une expression } E' \text{ compatible minimale avec } E \text{ telle que } w \in L(E').$$

Démonstration. Soit w un mot de $L(E')$. Puisque E' est une expression compatible avec E, selon le Lemme 3.6, $w \in L(E)$.

Soit w un mot de $L(E)$. Selon le Lemme 3.6, il existe une expression $E' = E'_{1\cdots n}$ telle que $w \in L(E')$. Si E' n'est pas une expression compatible minimale, on peut alors transformer un élément $\overline{E_k}$ en E_k dans E' et obtenir une expression E'' compatible avec E. Et selon le Lemme 3.7, le langage $L(E')$ est inclus dans le langage $L(E'')$. En itérant cette étape, nous obtenons une expression E' compatible minimale de E telle que w soit un mot de $L(E')$. □

Corollaire 3.2 *Soit $E = \text{——}_B(E_{1,n})$ une multi-barres et soit $\mathcal{E}$ l'ensemble des expressions compatibles minimales de E. Alors :*

$$L(E) = \bigcup_{E' \in \mathcal{E}} L(E').$$

Le nombre d'expressions compatibles minimales d'une multi-barres peut s'avérer exponentiel. Cette complexité est montrée dans la suite de cette section.

Proposition 3.2 *Pour tout entier n supérieur à 3, il existe une multi-barres E de largeur n telle que le cardinal de l'ensemble des expressions compatibles minimales de E soit exponentiel en fonction de n.*

Considérons la famille d'expressions $E_n = \overline{}_{B_n}(F_{1,n})$ définies par $B_n = ((1,2),(2,3),\ldots,(n-1,n))$ pour tout entier n supérieur ou égal à 3 et $F_k = (a_k + \varepsilon)$ pour $k \in [\![1,n]\!]$. Considérons les expressions E_3 et E_4 :

$E_3 = \overline{}_{B_3}(a_1+\varepsilon, a_2+\varepsilon, a_3+\varepsilon)$ avec $B_3 = ((1,2),(2,3))$,

$E_4 = \overline{}_{B_4}(a_1+\varepsilon, a_2+\varepsilon, a_3+\varepsilon, a_4+\varepsilon)$ avec $B_4 = ((1,2),(2,3),(3,4))$.

L'ensemble des expressions compatibles minimales de E_3 et de E_4 sont respectivement :

$$\mathcal{E}_3 = \{\overline{F_1\,F_2\,\overline{F_3}}, F_1\,\overline{F_2\,F_3}\},$$
$$\mathcal{E}_4 = \{\overline{F_1\,F_2\,\overline{F_3\,F_4}}, F_1\,\overline{F_2\,F_3\,\overline{F_4}}, F_1\,\overline{F_2}\;\overline{F_3\,F_4}\}.$$

Nous classifions ces expressions compatibles minimales en trois types, en fonction de leurs trois dernières valeurs. Les expressions de type 1 se terminent par $\overline{F_{n-2}\,F_{n-1}\,\overline{F_n}}$. Les expressions de type 2 se terminent par $\overline{F_{n-2}}\;\overline{F_{n-1}\,F_n}$. Les expressions de type 3 se terminent par $F_{n-2}\,\overline{F_{n-1}\,F_n}$. Nous générons alors des expressions compatibles minimales de E_{n+1} à partir des expressions compatibles minimales de E_n. Remarquons que nous ne montrons pas que nous les générons toutes, nous montrons que l'ensemble généré, inclus dans l'ensemble des expressions compatibles minimales de E_{n+1}, est de cardinal exponentiel en fonction de n.

Les expressions compatibles minimales de E_3 sont de deux types : $\overline{F_1\,F_2\,\overline{F_3}}$ est de type 1, $F_1\,\overline{F_2\,F_3}$ est de type 3. Pour E_4, les expressions compatibles minimales sont de trois types : $\overline{F_1\,F_2\,\overline{F_3\,F_4}}$ est de type 3, $F_1\,\overline{F_2\,F_3\,\overline{F_4}}$ est de type 1, et $F_1\,\overline{F_2}\;\overline{F_3\,F_4}$ est de type 2.

Une expression de type 3 s'obtient en ajoutant F_{n+1} à une expression de type 1. Une expression de type 2 s'obtient en ajoutant F_{n+1} à une expression de type 3 puis en remplaçant F_n par $\overline{F_n}$. Une expression de type 1 s'obtient soit en ajoutant $\overline{F_{n+1}}$ à une expression de type 2, soit en ajoutant $\overline{F_{n+1}}$ à une expression de type 3.

Nous définissons alors trois ensembles, notés respectivement $\mathrm{e}_1(n)$, $\mathrm{e}_2(n)$ et $\mathrm{e}_3(n)$, contenant des expressions compatibles minimales de E_n respectivement de type 1, 2 et 3. Ces ensembles sont récursivement calculables comme suit :

$$\mathrm{e}_1(n+1) = \{F'_{1\ldots n}\,\overline{F_{n+1}}) \mid F'_{1\ldots n} \in \mathrm{e}_2(n) \cup \mathrm{e}_3(n)\},$$
$$\mathrm{e}_2(n+1) = \{F'_{1\ldots n-1}\,\overline{F_n}\,F_{n+1}) \mid \{F'_{1\ldots n-1}F_n \in \mathrm{e}_3(n)\},$$
$$\mathrm{e}_3(n+1) = \{F'_{1\ldots n}F_{n+1}) \mid F'_{1\ldots n} \in \mathrm{e}_1(n)\}.$$

La minimalité et la compatibilité des expressions engendrées par ces formules sont vérifiées par induction sur n. Les rangs $n = 3$ et $n = 4$ sont corrects. Fixons alors un rang $n \geq 4$ vérifiant ces propriétés. Pour $\mathrm{e}_1(n+1)$, les expressions sont engendrées à partir des expressions $F'_{1\cdots n-2}\,\overline{F_{n-1}\,F_n}$ compatibles minimales de E_n. En ajoutant $\overline{F_{n+1}}$, la compatibilité est conservée pour l'ajout de la nouvelle barre $(n, n+1)$ lors du passage de B_n à

B_{n+1}. Par hypothèse d'induction et par définition de $e_1(n+1)$, toute substitution d'une sous-expression $\overline{E_k}$ par E_k fait perdre la compatibilité, d'où la minimalité de ces expressions compatibles. Le raisonnement est identique pour les ensembles $e_2(n+1)$ et $e_3(n+1)$, puisque toute barre de B_n est vérifiée par hypothèse d'induction, et que la barre $(n, n+1)$ ajoutée pour passer de B_n à B_{n+1} est vérifiée par les modifications engendrées. Ainsi, chacun des ensembles $e_k(n+1)$ pour $k = \{1,2,3\}$ est inclus dans l'ensemble des expressions compatibles minimales de E_{n+1}.

Calculons alors le cardinal de ces ensembles. Par définition :

$$\begin{aligned} \#e_1(n) &= \#e_2(n-1) + \#e_3(n-1), \\ \#e_2(n) &= \#e_3(n-1), \\ \#e_3(n) &= \#e_1(n-1). \end{aligned}$$

Et ainsi :

$$\#e_2(n) = \#e_3(n-1) = \#e_1(n-2).$$

D'où :

$$\#e_1(n) = \#e_1(n-2) + \#e_1(n-3).$$

Cette suite est la *suite de Padovan* (séquence *A000931* de l'OEIS) :

Définition 3.7 *La suite de Padovan* P *est définie par :*

$$P(n) = \begin{cases} 1 & \text{si } n = 0 \text{ ou } n = 1 \text{ ou } n = 2, \\ P(n-3) + P(n-2) & \text{sinon.} \end{cases}$$

De plus, $P(n) \approx \frac{p^n}{4}$, où p est le *nombre d'argent* (ou *nombre plastique*), unique solution réelle de l'équation $x^3 - x - 1 = 0$ ($p \approx 1.3247$).

Remarquons alors que pour $k \geq 3$, $\#e_1(k) = P(k-2)$. La Proposition 3.2 est alors démontrée. Cependant, bien que le nombre d'expressions compatibles minimales soit exponentiel, on ne peut rien affirmer quant à la largeur d'une expression simple dénotant le langage, puisque des factorisations sont possibles.

3.5.2 Multi-Barres et Expressions Linéarisées

Selon la Définition 3.2, le langage dénoté par une multi-barres est défini par un ensemble de contraintes sur les facteurs devant toutes être vérifiées. On peut alors tenter de calculer le langage dénoté à partir d'intersections de langages. Il suffirait de décomposer un opérateur de multi-barres comme une intersection de barres simples. Mais cela ne suffit pas, comme le montre l'exemple suivant :

Exemple 3.10 *Considérons les trois expressions E_1, E_2 et E_3 suivantes :*

$$\begin{aligned} E_1 &= \text{—}_{(1,2),(2,3)}\big((a+\varepsilon),(aa+\varepsilon),(a+\varepsilon)\big) = \overline{\overline{(a+\varepsilon)(aa+\varepsilon)}(a+\varepsilon)} \\ E_2 &= \text{—}_{(1,2)}\big((a+\varepsilon),(aa+\varepsilon),(a+\varepsilon)\big) = \overline{(a+\varepsilon)(aa+\varepsilon)}\,(a+\varepsilon) \\ E_3 &= \text{—}_{(2,3)}\big((a+\varepsilon),(aa+\varepsilon),(a+\varepsilon)\big) = (a+\varepsilon)\,\overline{(aa+\varepsilon)(a+\varepsilon)} \end{aligned}$$

Par définition, le langage $L(E_1)$ est $\{aaaa, aaa, aa\}$, alors que les langages $L(E_2)$ et $L(E_3)$ sont identiques et sont $\{aaaa, aaa, aa, a\}$. Les langages $L(E_1)$ et $L(E_2) \cap L(E_3)$ sont différents.

Un mot peut admettre plusieurs décompositions, chacune vérifiant une barre, et cela n'implique pas qu'il existe une décomposition vérifiant toutes les barres. Cependant, tout opérateur de multi-barres est compatible avec la linéarisation. Le langage d'une multi-barres peut alors se calculer à partir de son expression linéarisée.

Lemme 3.9 *Soit $E = \ —_B(E_{1,n})$ une multi-barres linéaire. Alors :*

$$L(E) = \begin{cases} \bigcap_{(i,f)\in B} L(E_{1\cdots i-1} \cdot \overline{E_{i\cdots f}} \cdot E_{f+1\cdots n}) & \text{si } B \neq \emptyset, \\ L(E_{1\cdots n}) & \text{sinon.} \end{cases}$$

Démonstration. Si B est vide, l'égalité est trivialement démontrée.

Soit w un mot de $L(\ —_B(E_{1,n}))$. Par définition, il existe une décomposition (unique puisque $E_{1\cdots n}$ est linéaire) de w en $w_1 \cdots w_n$ telle que pour tout entier k de $[\![1, n]\!]$, $w_k \in L(E_k)$ et telle que pour toute barre (i, f) de B, $w_i \cdots w_f \neq \varepsilon$. Ainsi, pour toute barre (i, f) de B, w est un mot du langage $L(E_{1\cdots i-1} \cdot \overline{E_{i\cdots f}} \cdot E_{f+1\cdots n})$, et ainsi, w appartient au langage défini par $\bigcap_{(i,f)\in B} L(E_{1\cdots i-1} \cdot \overline{E_{i\cdots f}} \cdot E_{f+1\cdots n})$.

Réciproquement, un mot $w = w_1 \cdots w_n$ qui appartient au langage $L(E_{1\cdots i-1} \cdot \overline{E_{i\cdots f}} \cdot E_{f+1\cdots n})$ pour une barre (i, f) de B donnée admet une décomposition (unique encore une fois) telle que pour tout entier k de $[\![1, n]\!]$, $w_k \in L(E_k)$ et telle que $w_i \cdots w_f \neq \varepsilon$. Ainsi, pour un mot w appartenant à $\bigcap_{(i,f)\in B} L(E_{1\cdots i-1} \cdot \overline{E_{i\cdots f}} \cdot E_{f+1\cdots n})$, puisque sa décomposition $w_1 \cdots w_n$ est unique, tout facteur $w_i \cdots w_f$ avec (i, f) une barre de B est différent du mot vide ε. Et selon la Définition 3.2, w appartient à $L(E)$. □

Corollaire 3.3 *Soit $E = \ —_B(E_{1,n})$ une multi-barres n'utilisant que des opérateurs compatibles avec la linéarisation et soit $E^{\#} = \ —_B(E'_{1,n})$ l'expression linéarisée de E. Alors :*

$$L(E) = \mathrm{h}_E(L(\textstyle\bigcap_{(i,f)\in B} L(\ —_{(i,f)}(E'_{1,n})))).$$

3.5.3 Multi-Tildes et Sous-Listes Libres

Selon la Définition 3.4, un mot appartient au langage dénoté par une multi-tildes définie par une liste T de couples s'il est généré par une sous-liste libre de T. Considérons alors le cas des opérateurs de multi-tildes définis sur des listes libres.

Lemme 3.10 *Soit n un entier positif. Soit $T = ((i_k, f_k)_{k\in[\![1,\#T]\!]})$ une liste libre de $\mathcal{S}_n$. Soit $E = \ \sim_T(E_{1,n})$ une multi-tildes. Posons :*

$$L_T = L(E_{1\cdots i_1-1}) \cdot L(\widetilde{E_{i_1\cdots f_1}}) \cdot L(E_{f_1+1\cdots i_2-1})$$
$$\cdots L(\widetilde{E_{i_{\#T}\cdots f_{\#T}}}) \cdot L(E_{f_{\#T}+1\cdots n}).$$

L'expression E dénote le langage L_T.

Démonstration. Soit w un mot de $L(E)$. Selon la Définition 3.4, il existe une sous-liste $T' = ((i'_l, f'_l))_{l\in[\![1,\#T']\!]}$ de T telle que T' génère w. Selon la Définition 3.3, le mot w appartient au langage défini par :

$$L(E_{1\cdots i'_1-1}) \cdot \{\varepsilon\} \cdot L(E_{f'_1+1\cdots i'_2-1}) \cdots \{\varepsilon\} \cdot L(E_{f'_{\#T'}+1\cdots n}),$$

langage lui même inclus dans le langage L_T. Par conséquent, $w \in L_T$.

Soit w un mot de L_T. Par définition, w admet une décomposition :

$$(w_1 \cdots w_{i_1-1}) \cdot (w_{i_1} \cdots w_{f_1}) \cdots (w_{f_{\#T}+1} \cdots w_n)$$

telle que pour tout couple (k, k') de l'ensemble $\{(1, i_1 - 1), \ldots, (f_{\#T+1}, n)\}$, soit $w_k \cdots w_{k'} \in L(E_{k\cdots k'})$, soit $w_k \cdots w_{k'} = \varepsilon$ et $(k, k') \in T$. Considérons la liste T' définie par :

$$T' = \big((i', f') \mid w_{i'} \cdots w_{f'} = \varepsilon \wedge (i', f') \in T\big).$$

Par construction, la liste T' est libre et $T' \subset T$. Selon la Définition 3.3, la liste T' génère le mot w. Finalement, selon la Définition 3.4, w est dans $L(E)$. □

Corollaire 3.4 *Soit n un entier positif, T une liste libre de $\mathcal{S}_n$ et $E = \sim_T(E_{1,n})$ une multi-tildes. Soit $\mathcal{W}$ l'ensemble des mots générés par T. Alors :*

$$\mathcal{W} \subset L(E).$$

L'ensemble des mots généres par une liste libre est inclus dans le langage dénoté par une multi-tilde défini sur cette liste. Par conséquent, le langage dénoté par une multi-tildes définie par une liste T quelconque est l'union des langages dénotés par des multi-tildes définis sur des sous-listes libres de T.

Lemme 3.11 *Soit $E = \sim_T(E_{1,n})$ une multi-tildes. Soit $\mathcal{T}$ l'ensemble des sous-listes libres de T. Alors :*

$$L(E) = \bigcup_{T'\in\mathcal{T}} L(\sim_{T'}(E_{1,n})).$$

Démonstration. Soit w un mot de $L(E)$. Selon la Définition 3.4, il existe une sous-liste libre T' de T qui génère w. Selon le Corollaire 3.4, w est dans $L(\sim_{T'}(E_{1,n}))$. Par conséquent, puisque $T' \in \mathcal{T}$, le mot w appartient à $\bigcup_{T'\in\mathcal{T}} L(\sim_{T'}(E_{1,n}))$.

Soit w un mot de $\bigcup_{T'\in\mathcal{T}} L(\sim_{T'}(E_{1,n}))$. Il existe $T' = ((i'_l, f'_l))_{l\in[\![1,\#T']\!]}$ dans $\mathcal{T}$ tel que w soit dans $L(\sim_{T'}(E_{1,n}))$. Selon la Définition 3.4, il existe une sous-liste libre T'' de T' telle que T'' génère w. La liste T'' est par définition une sous-liste libre de T. Par conséquent, selon la Définition 3.4, w est dans $L(E)$. □

Corollaire 3.5 *Soit n un entier positif, $E_{1,n}$ une liste d'expressions rationnelles et $E = \sim_T(E_{1,n})$ une multi-tildes. Alors :*

le langage dénoté par E est rationnel.

Le calcul de toutes les sous-listes libres de T est suffisant, mais pas nécessaire. En effet, les langages définis par une sous-liste libre d'une liste de tildes admettent une hiérarchie en terme d'inclusion. Le calcul du langage ne nécessite alors que l'utilisation de sous-listes particulières :

Définition 3.8 *Soit n un entier positif. Soit S une liste de $\mathcal{S}_n$. Une liste libre $S' \subset S$ est* maximale *si et seulement si :*

$$\forall (i,f) \in S \setminus S', S' \cup (i,f) \text{ n'est pas une liste libre.}$$

Lemme 3.12 *Soit n un entier positif. Soit S_1, S_2 deux listes libres de $\mathcal{S}_n$. Soit $E_{1,n}$ une liste d'expressions. Alors on a :*

$$S_1 \subset S_2 \Rightarrow L(\sim_{S_1}(E_{1,n})) \subset L(\sim_{S_2}(E_{1,n})).$$

Démonstration. Si w est un mot de $L(\sim_{S_1}(E_{1,n}))$, selon le Lemme 3.11, il existe une sous-liste S' de S_1 telle que $w \in L(\sim_{S'}(E_{1,n}))$. Puisque $S_1 \subset S_2$, alors $S' \subset S_2$. Ainsi, S' est une sous-liste de S_2 et selon le Lemme 3.11, $L(\sim_{S'}(E_{1,n})) \subset L(\sim_{S_2}(E_{1,n}))$. Par conséquent, w est un mot de $L(\sim_{S_2}(E_{1,n}))$. □

Corollaire 3.6 *Soit $E = \sim_T(E_{1,n})$ une multi-tildes et soit $\mathcal{T}_{max}$ l'ensemble des sous-listes libres maximales de T. Alors :*

$$L(E) = \bigcup_{T' \in \mathcal{T}_{max}} L(\sim_{T'}(E_{1,n})).$$

3.6 ... Définissant des Expressions Augmentées

Nous définissons alors de nouvelles expressions utilisant la famille des opérateurs de multi-barres et de multi-tildes qui sont rationnels et compatibles avec la notion de linéarisation, selon le Lemme 3.5, le Corollaire 3.1 et le Corollaire 3.5.

Définition 3.9 *Une* Expression Rationnelle Augmentée par Barres et Tildes *(**ERABT**) sur un alphabet Σ est définie inductivement comme suit :*

$$E = \emptyset, \quad E = \varepsilon, \quad E = a,$$
$$E = (F + G), \quad E = (F \cdot G), \quad E = (F^*),$$
$$E = (\text{—}_B(E_{1,n})), \quad E = (\sim_T(E_{1,n})),$$

avec a une lettre de Σ, F et G deux ERABT,
$E_{1,n}$ une liste d'ERABT, B et T des listes de $\mathcal{S}_n$.

3.7 Conclusion

L'ajout et l'élimination du mot vide dans le langage dénoté par un facteur d'une concaténation d'expressions permet de définir le caractère optionnel ou non de ce facteur. Pour modéliser ces notions, nous avons introduit deux nouveaux opérateurs unaires : la tilde crée un degré de liberté, en transformant un facteur en facteur effaçable, alors que la barre, son opérateur dual, crée une contrainte, en forçant un facteur à ne pas être effacé. La rationalité de ces opérateurs ainsi que leur compatibilité avec la linéarisation nous permet de calculer simplement un automate fini à partir de toute expression utilisant ces opérateurs. Nous avons également défini les multi-barres et les multi-tildes, nouvelles familles d'opérateurs étendant le pouvoir de la barre et de la tilde en augmentant l'arité d'application. Ces opérateurs sont rationnels et compatibles avec les notions de linéarisation. Ces propriétés nous permettront de calculer un automate par l'extension de la méthode de Glushkov dans le Chapitre 4. Nous présenterons également le pouvoir de factorisation des ERABT dans le Chapitre 7.

Chapitre 4

Formes Normalisées

C'est la fonction qui modifie la forme.
Konrad Lorenz

La forme, c'est le fond qui remonte à la surface.
Victor Hugo

Sommaire

Dans ce chapitre, nous présentons comment obtenir un automate reconnaissant le langage dénoté par une expression utilisant des opérateurs de multi-barres et de multi-tildes. Pour construire cet automate, nous utilisons une forme particulière, la forme normalisée. La Section 4.1 présente des notations communes aux multi-barres et aux multi-tildes. La Section 4.2 illustre la nécessité de la forme normalisée, due en particulier aux notions d'utilité et de traversabilité qui sont définies dans la Section 4.3. Nous montrons comment calculer cette forme normalisée à partir d'une ERABT quelconque dans la Section 4.4, où sont décrites les propriétés qui rendent possible le calcul des fonctions de Glushkov d'une ERABT en forme normalisée. Ce calcul est explicité dans la Section 4.5.

4.1 Introduction

Le chapitre précédent a introduit un nouveau type d'expressions, les ERABT, utilisant les deux nouvelles familles d'opérateurs, les multi-barres et les multi-tildes. Dans la suite de ce chapitre, toute expression étant soit une multi-tildes, soit une multi-barres est appelée *expression à contraintes*. Toute expression à contraintes E est représentée par $E = \text{---}_S(E_{1,n})$, avec S une liste de $\mathcal{S}_n$ et $E_{1,n}$ une liste d'ERABT.

Soit n un entier supérieur à 0 et soit S une liste de $\mathcal{S}_n$. Considérons l'ERABT $E = \text{---}_S(E_{1,n})$. Soit $e = (i, f)$ un couple de S ; le *rang de e* est l'intervalle sur lequel il est défini et est noté $\text{Rangs}(e) = [\![i, f]\!]$. Soit k un entier de $[\![1, n]\!]$. La *liste des couples couvrant l'expression* E_k est notée $\text{Elements}(k)$ et est définie par $\text{Elements}(k) = \big((i, f) \in S \mid k \in [\![i, f]\!]\big)$. Si ---_S est un opérateur de multi-barres (resp. de multi-tildes), la *liste des barres* (resp. la *liste des tildes*) *couvrant l'expression* E_k est définie par :
$\text{Barres}(k) = \big((i, f) \in S \mid k \in [\![i, f]\!]\big)$ (resp. $\text{Tildes}(k) = \big((i, f) \in S \mid k \in [\![i, f]\!]\big)$).
La liste S est *chevauchante* si et seulement si tout élément de S est chevauchant. La liste S est *continue* si et seulement si pour tout $k \in [\![1, n-1]\!]$, $\text{Elements}(k) \cap \text{Elements}(k+1) \neq \emptyset$. L'arité de la liste S est *minimale* si et seulement si $\forall k \in [\![1, n-1]\!]$, $\text{Elements}(k) \neq \text{Elements}(k+1)$. La liste $S = \big((i_k, f_k) \mid k \in [\![1, \#S]\!]\big)$ *recouvre* $[\![1, n]\!]$ si et seulement si elle est libre, si $i_1 = i$, si $f_{\#S} = f$ et si pour tout entier k de $[\![1, \#S - 1]\!]$, $f_k = i_{k+1} - 1$.

Nous étendons le quotient des expressions rationnelles simples aux ERABT. Si une expression sous un opérateur à contraintes est égale au mot vide, alors l'arité de l'opérateur peut se voir diminuer, puisque cette expression disparait lors de la concaténation implicite du calcul du langage. La liste de couples définissant l'opérateur peut se voir modifier.

Exemple 4.1 *Considérons l'expression* $E = \text{---}_{(1,2),(2,3)}\big(a + \varepsilon, \varepsilon, b + \varepsilon\big)$ *dont la représentation graphique est* $\overline{\overline{(a + \varepsilon)}\ \overline{(\varepsilon)}}\ \overline{(b + \varepsilon)}$ *. La seconde expression de la liste est égale au mot vide. Indépendamment du type d'opérateur utilisé (multi-tildes ou multi-barres), l'expression E est équivalente à* $E' = \text{---}_{(1,1),(2,2)}\big(a + \varepsilon, b + \varepsilon\big)$ *représentée par* $\overline{(a + \varepsilon)}\ \overline{(b + \varepsilon)}$ *puisque* $L(\varepsilon)$ *est nécessairement absorbée par la concaténation implicite des langages.*

Plus formellement, nous définissons le quotient suivant sur les ERABT. Soit n un entier supérieur à 0. Soit $E_{1,n}$ une liste d'ERABT, soit S une liste de $\mathcal{S}_n$ et soit $E = \text{---}_S(E_{1,n})$ une ERABT.

Supposons qu'il existe un entier k de $[\![1, n]\!]$ tel que $E_k = \varepsilon$.

(I) Supposons que le couple (k, k) soit dans S. Si E est une multi-barre, alors $E \sim \emptyset$. Si E est une multi-tildes, $E \sim \text{---}_{S \setminus \{(k,k)\}}(E_{1,n})$.

(II) Supposons $n \geq 2$. Nous définissons le quotient $E \sim \text{---}_{S'}(E'_{1,n})$ avec :

$$
\begin{array}{l}
E'_{k'} = \left\{ \begin{array}{l} E'_{k'} = E_{k'} \quad \text{si } k' \in [\![1, k-1]\!], \\ E'_{k'} = E_{k'+1} \text{ si } k' \in [\![k, n-1]\!], \end{array} \right. \\
S'_1 = \{(i,f) \in S \mid f < k\}, \\
S'_2 = \{(i-1, f-1) \mid (i,f) \in S \ \wedge \ i > k\}, \\
S'_3 = \{(i, f-1) \mid (i,f) \in S \ \wedge \ i \leq k \ \wedge \ k \leq f\}, \\
S' = S'_1 \cup S'_2 \cup S'_3.
\end{array}
$$

(III) Si $n = 1$, supposons $E_1 = \varepsilon$. Si ---_S est un opérateur de multi-barres vide ou s'il est un opérateur de multi-tildes, alors on a $E \sim \varepsilon$. Si ---_S est un opérateur de multi-barres non-vide, alors $(1,1)$ est dans S et le cas **(I)** s'applique.

Nous supposons alors que pour tout entier n supérieur à 0, pour toute ERABT $E = \text{---}_S(E_{1,n})$ et pour tout k de $[\![1, n]\!]$, le langage $L(E_k)$ est différent de $\{\varepsilon\}$.

4.2 Pourquoi une Forme Normalisée

Les représentations graphiques utilisées peuvent amener à une collision au niveau des ERABT. En effet, deux ERABT différentes peuvent avoir la même représentation graphique, comme le montre l'Exemple 4.2. Ici, nous souhaitons mettre en place une forme particulière sur les ERABT afin que deux expressions possédant la même représentation graphique puissent se ramener à une même expression sous cette forme.

Exemple 4.2 *Soit $T_1 = ((1,2),(2,2),(3,3),(3,4))$ une liste de $\mathcal{S}_4$, et soit $T_2 = ((1,2),(2,2))$ et $T_3 = ((1,1),(1,2))$ deux listes de $\mathcal{S}_2$. Considérons alors les deux ERABT $E_1 = \sim_{T_1}(a,b,c,d)$ et $E_2 = \sim_{T_2}(a,b) \cdot \sim_{T_3}(c,d)$ représentées graphiquement par $E_1 = \overbrace{a\ b}\ \overbrace{c\ d}$ et $E_2 = \overbrace{a\ b}\ \overbrace{c\ d}$. Remarquons que les représentations graphiques sont identiques, bien que les expressions soient syntaxiquement différentes.*

Nous souhaitons également que la forme particulière mise en place soit telle que deux expressions à contraintes sous cette forme, basées sur deux opérateurs distincts mais de même type et appliqués sur une même liste d'expressions ne dénotent pas le même langage. Cette propriété est appelée *utilité des couples* et sera développée dans la section suivante. Les deux exemples suivants montrent que deux opérateurs à contraintes distincts appliqués à une même liste d'ERABT peuvent définir le même langage.

Exemple 4.3 *Soit $B = ((1,2),(2,3))$ et soit $B' = ((1,2),(1,3),(2,3))$ deux listes de $\mathcal{S}_3$. Soit $E_{1,3} = ((a+\varepsilon),(b+\varepsilon),(c+\varepsilon))$ une liste d'ERABT. Soit $E = \text{——}_B(E_{1,3})$ et soit $E' = \text{——}_{B'}(E_{1,3})$ deux ERABT représentées graphiquement comme suit :*

$$E = \overline{\overline{(a+\varepsilon)(b+\varepsilon)}(c+\varepsilon)}\ ,$$
$$E' = \overline{\overline{\overline{(a+\varepsilon)(b+\varepsilon)}}(c+\varepsilon)}\ .$$

On peut vérifier que $L(E) = L(E')$ et qu'ainsi la présence de la barre $(1,3)$ de B' n'est pas nécessaire.

Exemple 4.4 *Soit $E_1 = \widetilde{a\ \widetilde{b}\ \widetilde{c}\ d}$ et $E_2 = \widetilde{a\ \widetilde{b}}\ \widetilde{\widetilde{c}\ d}$ deux ERABT, respectivement définies sur la liste de tildes $T_1 = ((1,2),(2,2),(2,3),(3,3),(3,4))$ et sur la liste de tildes $T_2 = ((1,2),(2,2),(3,3),(3,4))$. On peut vérifier que les langages $L(E_1)$ et $L(E_2)$ sont égaux et qu'ainsi la présence de la tilde $(2,3)$ de T_1 n'est pas nécessaire.*

4.3 Utilité des Éléments

La notion d'utilité d'un élément e d'un opérateur à contraintes est basée sur la réflexion suivante : si e ne modifie pas le langage, pourquoi ne pas l'éliminer ? Puisque les tildes et les barres ne modifient le langage dénoté par une expression qu'en ajoutant ou en éliminant le mot vide de certains facteurs, la question de l'existence d'un mot peut être considérée comme secondaire par rapport à la traversabilité d'intervalles.

Remarquons que pour une multi-barres $E = \overline{\ \ }_B(E_{1,n})$, si pour un entier k de $[\![1,n]\!]$, l'expression E_k est telle que $L(E_k) = \emptyset$, alors le langage $L(E)$ est $\emptyset$. En effet, par définition du langage des multi-barres, $L(E)$ est inclus dans $L(E_{1\cdots n})$. Cette remarque n'est pas valable pour les multi-tildes. En effet, pour l'expression $F = \widetilde{(a)\quad(\emptyset)\quad(b)}$, la présence d'une occurence de l'ensemble vide dans la liste d'expressions ne réduit pas le langage $L(F)$ à $\emptyset$. En effet le langage d'une multi-tildes est égal à l'union des langages dénotés par les expressions formées à partir des sous-listes libres ; on a donc :

$$L(F) = L(\widetilde{a\cdot\emptyset}\cdot b) \cup L(a\cdot\widetilde{\emptyset\cdot b}) = L(a+b).$$

L'ensemble vide étant utile pour certaines expressions, nous le substituons par une lettre n'appartenant pas à l'alphabet de l'expression. Nous étendons en conséquence la notion de linéarisation à la $\emptyset$-linéarisation.

Définition 4.1 *Soit E une expression. L'expression E est* $\emptyset$-linéaire *si et seulement si elle est linéaire et si elle ne possède pas d'occurrences de $\emptyset$.*

Définition 4.2 *Une expression* $\emptyset$-linéarisée *$E^\flat$ d'une ERABT E est calculée en indexant indépendamment par deux alphabets distincts les occurrences de symboles et les occurrences de l'ensemble vide, puis en remplaçant dans E toute occurrence de symbole ou de l'ensemble vide par son indice.*

Exemple 4.5 *L'expression $\emptyset$-linéarisée de l'expression $E = (a\cdot b\cdot a + a\cdot\emptyset)\cdot b\cdot b\cdot\emptyset$ est*

$E^\flat = (1 \cdot 2 \cdot 3 + 4 \cdot I) \cdot 5 \cdot 6 \cdot II$, *avec les ensembles* $\mathrm{Pos}(E) = \{1,2,3,4,5,6\}$ *et* $\mathrm{Pos}_\emptyset(E) = \{I, II\}$.

Le morphisme associé est $\mathrm{h}_E : (\mathrm{Pos}(E) \cup \mathrm{Pos}_\emptyset(E))^* \to (\Sigma_E \cup \{\emptyset\})^*$ où $\emptyset$ représente le caractère associé à l'ensemble vide. Remarquons que $L(E^\flat) \neq \emptyset$ et que toute expression $\emptyset$-linéarisée est $\emptyset$-linéaire. La compatibilité avec la linéarisation s'étend à la $\emptyset$-linéarisation très simplement et ces deux notions sont équivalentes.

Définition 4.3 *Soit $\mathcal{O}$ un ensemble d'opérateurs. Les opérateurs de $\mathcal{O}$ sont* compatibles avec la $\emptyset$-linéarisation *si et seulement si pour toute expression E utilisant ces opérateurs, en notant $E^\flat$ l'expression $\emptyset$-linéarisée de E, on a :*

$$L(E) = \mathrm{h}_E(L(E^\flat) \cap (\mathrm{Pos}(E))^*).$$

Lemme 4.1 *Soit $\mathcal{O}$ un ensemble d'opérateurs. Les deux propositions suivantes sont équivalentes :*

(1) *Les opérateurs de $\mathcal{O}$ sont compatibles avec la $\emptyset$-linéarisation,*

(2) *Les opérateurs de $\mathcal{O}$ sont compatibles avec la linéarisation.*

Démonstration. Soit E une expression utilisant des opérateurs de $\mathcal{O}$, $E^\#$ l'expression linéarisée de E et $E^\flat$ l'expression $\emptyset$-linéarisée de E. Par construction, $L(E^\flat) \cap (\mathrm{Pos}(E))^* = L(E^\#)$. □

Si un couple ne modifie pas le langage d'une expression $\emptyset$-linéarisée d'une expression E, alors ce couple ne modifie pas le langage de E. La $\emptyset$-linéarité nous permet ainsi de donner une définition formelle de l'utilité d'un élément pour un opérateur à contraintes.

Définition 4.4 *Soit $E = \text{---}_S(E_{1,n})$ une ERABT $\emptyset$-linéaire. Un élément e de S est* utile *pour E si et seulement si :*

$$L(\text{---}_S(E_{1,n})) \neq L(\text{---}_{S\setminus\{e\}}(E_{1,n})).$$

La liste S est utile *pour E si tout élément de S est utile pour E.*

Nous détaillons plus précisément la notion d'utilité pour les opérateurs de multi-barres et de multi-tildes dans les sous-sections suivantes.

4.3.1 Utilité des Barres

L'action d'une barre $b = (i, f)$ d'une liste B sur une liste d'expressions $E_{1,n}$ est d'éliminer le mot vide du langage $L_{(i,f)}$ dénoté par l'expression $E_{i\cdots f}$. Ainsi, pour que b soit utile, il faut que le langage $L_{(i,f)}$ contienne le mot vide. Il faut également que l'action de cette barre n'ait pas été devancée par celle d'une barre plus courte qu'elle peut inclure. Plus précisément, la combinaison de ces deux propriétés nécessaires est également suffisante.

Lemme 4.2 *Soit $E = \ \text{—\!—}_B(E_{1,n})$ une ERABT $\emptyset$-linéaire et soit b une barre de B. La barre b est utile si et seulement si les deux propriétés suivantes sont vérifiées :*

- Pour tout entier k de Rangs(b)*, ε est dans $L(E_k)$,*

- Pour toute barre b' de B différente de b, Rangs$(b') \not\subset$ Rangs(b).

Démonstration. Trivialement, on sait que le langage $L(\ \text{—\!—}_B(E_{1,n}))$ est inclus dans $L(\ \text{—\!—}_{B\setminus\{b\}}(E_{1,n}))$, puisque si toute barre de B est vérifiée par un mot w, alors toute barre de $B \setminus \{b\}$ est vérifiée par w.

($\Rightarrow$) : (1) Supposons qu'il existe une barre $b' \in B \setminus \{b\}$ telle que Rangs$(b') \subset$ Rangs(b). Soit $w = w_1 \cdots w_n$ un mot de $L(\ \text{—\!—}_{B\setminus\{b\}}(E_{1,n}))$ tel que pour tout $k \in [\![1, n]\!]$, $w_k \in L(E_k)$. Par définition du langage d'une multi-barres, b' est vérifiée par w, et par conséquent il existe un entier k de Rangs(b') tel que $w_k \neq \varepsilon$. Puisque b' est inclus dans b, l'entier k appartient également à Rangs(b). Par conséquent, b est vérifiée par tout mot de $L(\ \text{—\!—}_{B\setminus\{b\}}(E_{1,n}))$ et finalement $L(\ \text{—\!—}_{B\setminus\{b\}}(E_{1,n})) \subset L(\ \text{—\!—}_B(E_{1,n}))$. Selon la Définition 4.4, la barre b est inutile. **(2)** Soit $w = w_1 \cdots w_k \cdots w_n$ un mot tel que pour tout k de $[\![1, n]\!]$, $w_k \in L(E_k)$. S'il existe un entier k de Rangs(b) tel que $\varepsilon \notin L(E_k)$, $w_k \neq \varepsilon$ et ainsi w vérifie b. Ainsi, puisque le langage $L(\ \text{—\!—}_{B\setminus\{b\}}(E_{1,n}))$ est inclus dans $L(E_{1\cdots n})$ tout mot $w \in L(\ \text{—\!—}_{B\setminus\{b\}}(E_{1,n}))$ vérifie b. Par conséquent, le langage $L(\ \text{—\!—}_{B\setminus\{b\}}(E_{1,n}))$ est inclus dans $L(\ \text{—\!—}_B(E_{1,n}))$. Selon la Définition 4.4, la barre b est inutile.

($\Leftarrow$) : Supposons que pour tout entier k de Rangs(b), ε est dans $L(E_k)$ et qu'il n'existe pas de barre b' de B différente de b telle que l'intervalle Rangs(b') soit inclus dans Rangs(b). Montrons alors que la barre b est utile pour E, et donc que les langages $L(\ \text{—\!—}_B(E_{1,n}))$ et $L(\ \text{—\!—}_{B\setminus\{b\}}(E_{1,n}))$ sont différents. Soit $w_1 \cdots w_n$ un mot tel que d'une part pour tout entier k qui n'est pas dans Rangs(b), le mot w_k appartienne à $L(\overline{E_k})$ et que d'autre part pour tout entier k de Rangs(b), le mot w_k soit égal au mot vide ε. Puisqu'il n'existe pas de barre b' différente de b telle que l'intervalle Rangs(b') soit inclus dans Rangs(b), toute barre b' de $B \setminus \{b\}$ est vérifiée puisque par construction de w, pour tout $k \notin$ Rangs(b), $w_k \in L(\overline{E_k})$. Et puisque pour tout $k \in$ Rangs(b), $\varepsilon \in L(E_k)$, selon la Définition 3.2, le mot w est dans $L(\ \text{—\!—}_{B\setminus\{b\}}(E_{1,n}))$. Puisque pour tout $k \in$ Rangs(b), $w_k = \varepsilon$, la barre b n'est pas vérifiée. Ainsi, le mot w n'appartient pas au langage $L(\ \text{—\!—}_B(E_{1,n}))$ et ainsi les langages $L(\ \text{—\!—}_B(E_{1,n}))$ et $L(\ \text{—\!—}_{B\setminus\{b\}}(E_{1,n}))$ sont différents. Selon la Définition 4.4, b est utile. □

4.3.2 Utilité des Tildes

L'action d'une tilde $t = (i, f)$ d'une liste T sur une liste d'expressions $E_{1,n}$ est d'ajouter le mot vide dans le langage $L_{(i,f)}$ dénoté par l'expression $E_{i\cdots f}$. Pour que le langage $L_{(i,f)}$ soit modifié par cette action, il faut que

$L_{(i,f)}$ ne contienne pas le mot vide. De plus, si une combinaison de tildes de T a déjà ajouté le mot vide dans $L_{(i,f)}$, la tilde t ne modifie pas le langage. On peut également considérer l'existence d'une combinaison de tildes plus courtes de T et de facteurs de $E_{1,n}$ dont le langage contient le mot vide, rendant l'action de la tilde inutile. Tous ces cas possèdent un point commun : le mot vide peut appartenir au langage dénoté par un facteur par l'action conjuguée de la traversabilité et d'une combinaison de tildes. Nous formalisons alors une nouvelle notion, celle de la (i,f)-traversabilité, et nous montrons qu'elle est due à la combinaison de tildes et de facteurs traversables.

Définition 4.5 *Soit $E = \sim_T(E_{1,n})$ une ERABT $\emptyset$-linéaire. Soit (i,f) un couple de $[\![1,n]\!]^2_{\leq}$. L'ERABT E est* (i,f)-traversable *si et seulement si :*

$$L(\overline{E_1}) \cdots L(\overline{E_{i-1}}) \cdot L(\overline{E_{f+1}}) \cdots L(\overline{E_n}) \subset L(E).$$

*Soit S une liste de $\mathcal{S}_n$. L'ERABT E est S-*traversable *si et seulement si :*
Pour tout (i,f) de S, E est (i,f)-traversable .

Lemme 4.3 *Soit $E = \sim_T(E_{1,n})$ une ERABT $\emptyset$-linéaire. Soit $(i,f) \in [\![1,n]\!]^2_{\leq}$. L'expression E est (i,f)-traversable si et seulement s'il existe une liste libre S satisfaisant les conditions suivantes :*
- S recouvre $[\![i,f]\!]$,
- Pour tout (i'_k, f'_k) de S, (i'_k, f'_k) est dans T ou ε est dans $L(E_{i'_k \cdots f'_k})$.

Démonstration. Posons $L' = L(\overline{E_1}) \cdots L(\overline{E_{i-1}}) \cdot L(\overline{E_{f+1}}) \cdots L(\overline{E_n})$ et $S = \big((i'_k, f'_k) \mid k \in [\![1, \#S]\!]\big)$.

(⇐) : Soit S une liste satisfaisant les conditions précédentes. Considérons la partition $(S_1 = S \cap T, S_2 = S \setminus S_1)$ de S. Puisque S est libre, S_1 est une sous-liste libre de T. Selon le Lemme 3.11, le langage $L(\sim_{S_1}(E_{1,n}))$ est inclus dans $L(E)$. De plus, puisque pour tout couple (i'_k, f'_k) de S_2 le langage dénoté par l'expression $E_{i'_k \cdots f'_k}$ contient le mot vide, les langages $L(\sim_S(E_{1,n}))$, $L(\sim_{S_1 \cup S_2}(E_{1,n}))$ et $L(\sim_{S_1}(E_{1,n}))$ sont égaux. Ainsi, le langage $L(\sim_S(E_{1,n}))$ est inclus dans $L(E)$. Soit w un mot de L'. Le mot w se décompose en $w_1 \cdots w_{i-1} \cdot w_{f+1} \cdots w_n$ tel que pour tout entier k de $[\![1, i-1]\!] \cup [\![f+1, n]\!]$, $w_k \in L(\overline{E_k})$. Trivialement, $\forall k \in [\![1, i-1]\!] \cup [\![f+1, n]\!]$, $w_k \in L(E_k)$, puisque par définition de la barre, $L(\overline{E_k}) \subset L(E_k)$. Selon la définition de S, $w \in L(\sim_S(E_{1,n}))$. Finalement, $L' \subset L(\sim_S(E_{1,n}))$ et $L(\sim_S(E_{1,n})) \subset L(E)$ et selon la Définition 4.5, l'expression E est (i,f)-traversable.

(⇒) : Supposons E (i,f)-traversable. Selon la Définition 4.5, $L' \subset L(E)$. Soit $w = w_1 \cdots w_{i-1} \cdot w_{f+1} \cdots w_n$ un mot de L' tel que pour tout entier k de $[\![1, i-1]\!] \cup [\![f+1, n]\!], w_k \in L(\overline{E_k})$. Puisque $L' \subset L(E)$, il existe une sous-liste libre S de T telle que $w \in L(\sim_S(E_{1,n}))$. Puisque le seul facteur

ε-maximal de w est $w_i \cdots w_f$, sans perte de généralité, on peut supposer que $S \subset [\![i, f]\!]^2_{\leq}$. Soit (i'_1, f'_1) (resp. (i'_s, f'_s)) le plus petit (resp. le plus grand) élément de la liste S et soit (i'_{k+1}, f'_{k+1}) le plus petit élément plus grand que (i'_k, f'_k). Considérons la liste V définie par :

$$V = \{(i, i'_1 - 1), (f'_s + 1, f)\} \cup \{(f'_k + 1, i'_{k+1} - 1) \mid (i'_k, f'_k), (i'_{k+1}, f'_{k+1}) \in T \cap [\![i, f]\!]^2_{\leq}\}.$$

Si $S = \emptyset$, posons $V = V \cup \{(i, f)\}$. Il n'existe pas de couple (i', f') de V tel que $i \leq i' \leq f' \leq f$ et $\varepsilon \notin L(E_{i'\dots f'})$. Sinon, selon le Lemme 3.11 le mot w ne serait pas dans $L(\sim_S(E_{1,n}))$. Deux cas sont à traiter : soit $L(E_{i'\dots f'})$ contient le mot vide pour tout couple (i', f') de V, soit pour tout $k \in [\![1, \#S - 1]\!]$, $f'_k = i'_{k+1}$, $i'_1 = i$ et $f'_{\#S} = f$. Dans ces deux cas, la liste $S \cup V$ de couples satisfait les propriétés du lemme. □

Corollaire 4.1 *Soit $E = \sim_T(E_{1,n})$ une ERABT $\emptyset$-linéaire. Soit $(i, f) \in [\![1, n]\!]^2_{\leq}$. L'expression E est (i, f)-traversable si et seulement si au moins une des deux propositions suivantes est vérifiée :*

- *$(i, f) \in T$,*
- *il existe $k \in [\![i, f - 1]\!]$ tel que E est (i, k)-traversable et $(k + 1, f)$-traversable.*

Montrons alors le lien entre les notions d'utilité d'une tilde (i, f) et la (i, f)-traversabilité d'une expression. Nous présentons tout d'abord l'équivalence entre le fait qu'un mot w possédant des facteurs ε- maximaux particuliers appartienne au langage d'une multi-tildes, et le fait que tout mot possédant ces mêmes facteurs ε- maximaux appartienne également au langage de la multi-tildes. Puis nous montrons l'équivalence entre la (i, f)-traversabilité et l'utilité d'une tilde (i, f).

Proposition 4.1 *Soit $E = \sim_T(E_{1,n})$ une ERABT $\emptyset$-linéaire. Soit $w = w_1 \cdots w_n$ un mot tel que pour tout k de $[\![1, n]\!]$, w_k est dans $L(E_k) \cup \{\varepsilon\}$. Soit S une liste libre telle que pour tout couple (i_k, f_k) de S, $w_{i_k} \cdots w_{f_k}$ est un facteur ε-maximal dans w. Les trois conditions suivantes sont équivalentes :*

(1) *$L(\sim_S(\overline{E_1}, \dots, \overline{E_n})) \subset L(E)$,*
(2) *E est S-traversable,*
(3) *$w \in L(E)$.*

Démonstration. Posons $E'_{1,n} = (\overline{E_1}, \dots, \overline{E_n})$ et $S = ((i_k, f_k))_{k \in [\![1, \#S]\!]}$. Remarquons que le langage $L(\sim_T(E'_{1,n}))$ est inclus dans $L(E)$ et que par construction de S, puisque pour tout k de $[\![1, \#S]\!]$, $w_{i_k} \cdots w_{f_k}$ est un facteur ε-maximal dans w, w est dans $L(\sim_S(E'_{1,n})) \subset L(\sim_S(E_{1,n}))$. Remarquons également que par construction de S, il n'existe pas deux tildes de S consécutives (pour tout (i, f) ,(i', f') de S, $i' \neq f + 1$) sinon cela contredirait la maximalité des ε-facteurs de w.

(1) ⇒ (2) : Selon le Lemme 3.10, pour tout couple (i, f) de S, puisque la liste $((i, f))$ est une sous-liste libre de S, le langage $L(\sim_{((i,f))}(E'_{1,n}))$

est inclus dans $L(\ \sim_S(E'_{1,n}))$ lui-même inclus dans $L(E)$. Ainsi, pour tout (i,f) de S, E est (i,f)-traversable et selon la Définition 4.5, E est S-traversable.

(2) ⇒ (3) : Selon le Lemme 4.3, pour tout couple (i_k, f_k) de S, il existe une liste S_k et une partition (S_{1k}, S_{2k}) de S_k telles que $S_{1k} = S_k \cap T$ et $S_{2k} = S_k \setminus S_{1k}$. Puisque S est libre, la liste $S' = \bigcup_{k\in[\![1,\#S]\!]} S_{1k}$ est libre et S' est inclus dans T. Selon le Lemme 3.11, $L(\ \sim_{S'}(E_{1,n})) \subset L(E)$. De plus, puisque pour tout couple (i_k, f_k) de $S'' = \bigcup_{k\in[\![1,\#S]\!]} S_{2k}$, le langage dénoté par $E_{i_k\cdots f_k}$ contient le mot vide et que pour tout couple $(i_{k'}, f_{k'})$ de S', $[\![i_{k'}, f_{k'}]\!] \cap [\![i_k, f_k]\!] = \emptyset$, les langages $L(\ \sim_S(E_{1,n}))$, $L(\ \sim_{S'\cup S''}(E_{1,n}))$ et $L(\ \sim_{S'}(E_{1,n}))$ sont égaux. Ainsi, le langage $L(\ \sim_S(E_{1,n}))$ est inclus dans $L(E)$ et puisque le langage $L(\ \sim_S(E'_{1,n}))$ est inclus dans $L(\ \sim_S(E_{1,n}))$, le mot w est dans $L(E)$.

(3) ⇒ (1) : Supposons w dans $L(E)$. Rappelons que w appartient également au langage $L(\ \sim_S(E'_{1,n}))$. Selon le Lemme 3.11, il existe une sous-liste libre S' de T telle que w appartient à $L(\ \sim_{S'}(E_{1,n}))$. Considérons un mot $w' = w'_1 \cdots w'_n$ de $L(\ \sim_S(E'_{1,n}))$ tel que $w' \notin L(\ \sim_{S'}(E_{1,n}))$. Puisque w' est un mot de $L(\ \sim_S(E'_{1,n}))$, selon la Définition 3.4 et qu'il n'existe pas de tildes consécutives dans S ni de mot vide dans les langages $L(E'_k)$ pour tout k de $[\![1,n]\!]$, pour tout facteur $w'_i \cdots w'_f$ ε-maximal de w', le couple (i,f) est dans S. Et puisque $w' \notin L(\ \sim_{S'}(E_{1,n}))$, il existe un facteur $w'_i \cdots w'_f$ ε-maximal tel que $\sim_{S'}(E_{1,n})$ ne soit pas (i,f)-traversable ; si ce n'était pas le cas, le Lemme 4.3 et la Définition 3.4 impliqueraient la présence de w' dans $L(\ \sim_{S'}(E_{1,n}))$. Par construction, puisque (i,f) est dans S, $w_i \cdots w_f$ est un facteur ε-maximal pour w. Puisque $\sim_{S'}(E_{1,n})$ n'est pas (i,f)-traversable, selon le Lemme 4.3, le mot w ne peut vérifier les critères d'appartenance au langage de la Définition 3.4. Nous aboutissons alors à une contradiction. Finalement, w' est dans $L(\ \sim_{S'}(E_{1,n})) \subset L(E)$. D'où $L(\ \sim_S(E'_{1,n})) \subset L(E)$. □

Corollaire 4.2 *Soit $E = \ \sim_T(E_{1,n})$ une ERABT $\emptyset$-linéaire. Soit (i,f) un couple de $[\![1,n]\!]^2_{\leq}$. Soit $w = w_1 \cdots w_n$ un mot tel que pour tout entier k de $[\![1,n]\!]$, $w_k = \varepsilon$ si k est dans $[\![i,f]\!]$, w_k est dans $L(E_k) \setminus \{\varepsilon\}$ sinon. Les deux conditions suivantes sont équivalentes :*

(1) *$w \in L(E)$,*

(2) *E est (i,f)-traversable.*

Lemme 4.4 *Soit $E = \ \sim_T(E_{1,n})$ une ERABT $\emptyset$-linéaire et soit $t = (i,f)$ une tilde de T. Les deux propositions suivantes sont équivalentes :*

(1) *La tilde t est utile pour E,*

(2) *L'expression $\sim_{T\setminus\{t\}}(E_{1,n})$ n'est pas (i,f)-traversable.*

Démonstration. Selon le Corollaire 4.1, puisque (i,f) est dans T, E est (i,f)-

traversable. Posons $E' = \sim_{T\setminus\{t\}}(E_{1,n})$.

(1 ⇐ 2) : Supposons E' non (i,f)-traversable. Selon la Proposition 4.1, on a $L(\sim_{((i,f))}(E_{1,n})) \not\subset L(E')$ alors que le langage $L(\sim_{((i,f))}(E_{1,n}))$ est inclus dans $L(E)$ puisque t est une tilde de T. Les expressions étant $\emptyset$-linéaires, les langages précédents ne sont pas vides. Par conséquent, les langages $L(\sim_{T}(E_{1,n}))$ et $L(\sim_{T\setminus\{t\}}(E_{1,n}))$ sont différents et selon la Définition 4.4, t est utile.

(1 ⇒ 2) : Supposons que E' est (i,f)-traversable. Soit S'' une liste de couples satisfaisant les conditions du Lemme 4.3. Pour toute sous-liste libre S de T, définissons S' comme suit :

$$S' = \begin{cases} S & \text{si } t \notin S, \\ S \setminus \{t\} \cup (S'' \cap T) & \text{sinon.} \end{cases}$$

Ainsi S' est une sous-liste libre de $T \setminus \{t\}$ et les langages $L(\sim_{S}(E_{1,n}))$ et $L(\sim_{S'}(E_{1,n}))$ sont égaux. Par conséquent, le langage $L(\sim_{T}(E_{1,n}))$ est inclus dans $L(\sim_{T\setminus\{t\}}(E_{1,n}))$. Selon le Lemme 3.12, $L(\sim_{T\setminus\{t\}}(E_{1,n}))$ est inclus dans $L(\sim_{T}(E_{1,n}))$. Par conséquent les langages $L(\sim_{T}(E_{1,n}))$ et $L(\sim_{T\setminus\{t\}}(E_{1,n}))$ sont égaux et selon la Définition 4.4, t n'est pas utile. □

4.4 Formes Normalisées

Dans cette section, nous définissons la Forme Normalisée d'une expression à contraintes, puis celle d'une ERABT. Nous montrons que si deux multi-barres (resp. multi-tildes) en Forme Normalisée sont définies par la même liste d'expressions, alors les langages sont égaux si et seulement si elles sont définies sur la même liste de barres (resp. de tildes).

Définition 4.6 *Soit $E = \dashrightarrow_{S}(E_{1,n})$ une ERABT $\emptyset$-linéaire. L'ERABT E est en* Forme Normalisée *(**FoNo**) si les quatre propositions suivantes sont vérifiées :*

- ***(1)** Tout élément de S est utile pour E,*
- ***(2)** Soit S est chevauchante, soit #S = 1,*
- ***(3)** La liste S est continue,*
- ***(4)** L'arité de S est minimale.*

Définition 4.7 *Soit E une ERABT $\emptyset$-linéaire. L'ERABT E est en FoNo si et seulement si toute sous-expression à contraintes de E est en Forme Normalisée.*

Définition 4.8 *Soit E une ERABT. L'expression E est en FoNo si et seulement si l'expression $\emptyset$-linéarisée de E est en FoNo.*

Exemple 4.6 *Soit $E_3 = \widetilde{(a \cdot \widetilde{b})} \cdot \widetilde{(\widetilde{c} \cdot d)}$ une ERABT dont l'arbre syntaxique est représenté sur la Figure 4.1. On peut vérifier que cette expression satisfait les conditions de la Définition 4.7 et qu'elle est équivalente aux expressions E_1 et E_2 de l'Exemple 4.4. Remarquons que les représentations graphiques de E_2 et E_3 sont*

très proches (identiques en éliminant les points de concaténation et les parenthèses), mais que leurs arbres syntaxiques sont distincts.

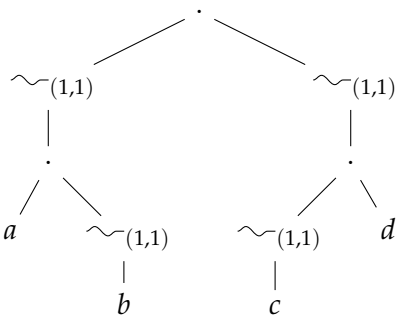


FIGURE 4.1 – *L'Expression E_3 est en FoNo.*

Proposition 4.2 *Soit n un entier positif, S_1 et S_2 deux listes de $\mathcal{S}_n$, et $F_{1,n}$ une liste d'ERABT en FoNo. Considérons deux ERABT $E_1 = \multimap_{S_1}(F_{1,n})$ et $E_2 = \multimap_{S_2}(F_{1,n})$ (resp. $E_1 = \sim_{S_1}(F_{1,n})$ et $E_2 = \sim_{S_2}(F_{1,n})$) telles que E_1 et E_2 soient $\emptyset$-linéaires et en FoNo. Les deux conditions suivantes sont équivalentes :*

(1) *$L(E_1) = L(E_2)$,*

(2) *$S_1 = S_2$.*

Démonstration. Montrons les deux sens de l'équivalence :

(2)⇒(1) Trivial.

(1)⇒(2) Supposons que $S_1 \neq S_2$. Alors il existe un élément $e = (i, f)$ dans la différence symétrique de S_1 et S_2 tel que $i - f + 1$ soit minimal. Sans perte de généralité, supposons e dans S_1. Nous allons traiter le cas des multi-barres et des multi-tildes séparément.

Cas des multi-barres : Supposons qu'il existe une barre e' de S_2 telle que $\text{Rangs}(e') \subsetneq \text{Rangs}(e)$ et montrons que nous aboutissons à une contradiction. Il y a deux cas. Supposons tout d'abord que e' soit dans S_1, ce qui implique que e est inutile, contradiction selon la Définition 4.6. Supposons alors que $e' \notin S_1$. Ce qui contredit le fait que e soit la plus courte barre de la différence symétrique de S_1 et S_2. Ainsi, il n'existe pas de barre e' dans S_2 telle que $\text{Rangs}(e') \subsetneq \text{Rangs}(e)$. Considérons un mot $w = w_1 \cdots w_{i-1} \cdot w_{f+1} \cdots w_n$ tel que $w_k \in L(F_k) \setminus \{\varepsilon\}$ pour tout k de $[\![1, i-1]\!] \cup [\![f+1, n]\!]$. Puisqu'il n'y a pas de barre e' telle que $\text{Rangs}(e')$ soit inclus dans $\text{Rangs}(e)$, toutes les barres de S_2 sont vérifiées par w, alors que w ne vérifie pas la barre e de S_1. Ainsi, $w \in L(E_2)$ et $w \notin L(E_1)$.

Cas des multi-tildes : La tilde e appartient à S_1. Selon le Lemme 4.3, l'expression E_1 est (i, f)-traversable. Supposons que l'expression E_2 soit (i, f)-traversable également. Si la tilde e est dans S_2, contradiction avec $t \in T_1 \triangle T_2$. Si $\varepsilon \in L(F_{i \cdots f})$, contradiction avec l'utilité de e pour E_1 selon le Lemme 4.4. S'il existe une liste de couples permettant à E_2 d'être (i, f)-

traversable, et s'il n'y a pas de tildes (i', f') de S_2 telle que $i \leq i' \leq f' < f$ ou $i < i' \leq f' \leq f$, on peut se ramener au cas précédent. Supposons qu'une telle liste de couples existe et qu'au moins un des couples soit une tilde de S_2. Soit une de ces tildes n'est pas incluse dans S_1 et on contredit la minimalité de la longueur de e, soit toutes ces tildes sont incluses dans S_1 et par conséquent e est inutile pour E_1. Ainsi, puisque E_1 est (i, f)-traversable alors que E_2 ne l'est pas, selon la Définition 4.5, leurs langages sont différents. □

Nous montrons maintenant que toute ERABT E admet une ERABT E' équivalente et en FoNo. La preuve de la Proposition 4.3 se réalise par l'application du Lemme 4.5, du Lemme 4.6 et du Lemme 4.7 qui fournissent une méthode de calcul de E' à partir de E.

Proposition 4.3 *Soit $E = \text{---}_S(E_{1,n})$ une ERABT. Alors il existe une ERABT en FoNo E' telle que les trois propositions suivantes soient vérifiées :*

(1) $L(E') = L(E)$,

(2) $\mathrm{Pos}(E') = \mathrm{Pos}(E)$,

(3) $\forall k \in \mathrm{Pos}(E)$, $\mathrm{h}_{E'}(k) = \mathrm{h}_E(k)$.

Lemme 4.5 *Soit $E = \text{---}_S(E_{1,n})$ une ERABT. Il existe une ERABT équivalente E' telle que pour toute sous-expression à contraintes $F' = \text{---}_{S'}(F'_{1,n})$ de E', soit $\#S' = 1$, soit S' ne contient pas d'élément surplombant.*

Démonstration. Si E et chacune de ses sous-expressions à contraintes ne contiennent pas d'éléments surplombants ou que $\#S = 1$, le lemme est vérifié. Supposons alors qu'une sous-expression à contraintes $F = \text{---}_R(F_{1,m})$ de E possède un élément surplombant.

Soit $e = (i, f)$ un élément surplombant de R.

(1) Si e n'est pas un élément inclus, R se décompose en :

$$R_1 = \{(i', f') \in R \mid f' < i\},$$
$$R_2 = \{(i', f') \in R \mid i \leq i' \leq f' \leq f\},$$
$$R_3 = \{(i', f') \in R \mid f < i'\}.$$

Considérons les deux sous-listes suivantes :

$$R'_2 = \{(i' - i + 1, f' - i + 1) \mid (i', f') \in R_2\} \setminus \{e\},$$
$$R'_3 = \{(i' - f, f' - f) \mid (i', f') \in R_3\}.$$

Alors les ERABT F et $F' = F'_1 \cdot F'_2 \cdot F'_3$ sont équivalentes, avec :

$$F'_1 = \text{---}_{R_1}(F_{1,i-1}),$$
$$F'_2 = \text{---}_{(1,1)}(\text{---}_{R'_2}(F_{i,f})),$$
$$F'_3 = \text{---}_{R'_3}(F_{f+1,m}).$$

(2) Si e est un élément inclus, R se décompose en :

$$R_2 = \{(i', f') \in R \mid i \leq i' \leq f' \leq f\},$$
$$R_1 = R \setminus R_2.$$

Considérons les deux sous-listes suivantes :

$$R'_2 = \{(i'-i+1, f'-i+1) \mid (i', f') \in R_2\} \setminus \{e\},$$
$$R'_1 = \{(i', f') \in R_1 \mid f' < i\}$$
$$\cup\{(i'-f+i, f'-f+i) \mid (i', f') \in R_1 \wedge f' < i\}.$$

Soit $F'_2 = \text{---}_{R'_2}(F_{i,f})$ et $F'_1 = \text{---}_{R'_1}(F_1, \ldots, F_{i-1}, \text{---}_{(1,1)}(F'_2), F_{f'+1}, \ldots, F_n)$ deux ERABT. Alors les ERABT F et F'_1 sont équivalentes et F'_1 ne contient pas d'élément surplombant.

Cette étape appliquée récursivement sur E (ainsi qu'à toutes ses sous-expressions à contraintes) permet de construire une expression vérifiant l'énoncé du lemme. □

Lemme 4.6 *Soit $E = \text{---}_S(E_{1,n})$ une ERABT. Alors il existe une ERABT E' équivalente à E telle que toute sous-expression à contraintes de E' soit continue.*

Démonstration. Si E et chacune de ses sous-expressions sont continues, E vérifie le lemme. Supposons alors qu'il existe une sous-expression à contraintes $F = \text{---}_R(F_{1,m})$ de E telle que R ne soit pas continu. Soit un entier k de $[\![1, m-1]\!]$ tel que $\text{Elements}(k) \cap \text{Elements}(k+1) = \emptyset$. Considérons les deux sous-listes suivantes :

$$R_1 = \{(i, f) \in R \mid f \leq k\},$$
$$R_2 = \{(i-k, f-k) \mid (i, f) \in R \ \wedge \ i > k\}.$$

Considérons F'_1 et F'_2 les expressions définies par :

$$F'_1 = \begin{cases} \text{---}_{R_1}(F_{1,k}) & \text{si } R_1 \neq \emptyset, \\ F_{1\cdots k} & \text{sinon,} \end{cases} \quad F'_2 = \begin{cases} \text{---}_{R_2}(F_{k+1,m}) & \text{si } R_2 \neq \emptyset, \\ F_{k+1\cdots m} & \text{sinon.} \end{cases}$$

On peut vérifier que F et $F' = F'_1 \cdot F'_2$ sont équivalentes.

Cette étape peut être appliquée récursivement, conduisant à une ERABT équivalente telle que toute sous-expression à contraintes de E' soit continue. □

Lemme 4.7 *Soit $E = \text{---}_S(E_{1,n})$ une ERABT. Alors il existe une ERABT équivalente E' telle que toute sous-expression à contraintes de E' soit d'arité minimale.*

Démonstration. Si E et chacune de ses sous-expressions à contraintes sont d'arité minimale, E vérifie le lemme. Supposons alors qu'il existe une sous-expression à contrainte $F = \text{---}_R(F_{1,m})$ de E telle que F ne soit pas d'arité minimale.

Soit k un entier de $[\![1, m-1]\!]$ tel que les ensembles $\text{Elements}(k)$ et $\text{Elements}(k+1)$ sont égaux. Soit $F' = \text{---}_{R'}(F'_{1,m-1})$ une ERABT avec :

$$\text{Pour tout entier } j \text{ de } [\![1, n-1]\!],\ F'_j = \begin{cases} F_j & \text{si } j < k, \\ F_j \cdot F_{j+1} & \text{si } j = k, \\ F_{j+1} & \text{si } k < j < m, \end{cases}$$

$$R' = \{(i, f) \mid (i, f) \in R \ \wedge \ f < k\}$$
$$\cup\{(i, f-1) \mid (i, f) \in R \ \wedge \ i \leq k < f\}$$
$$\cup\{(i-1, f-1) \mid (i, f) \in R \ \wedge \ k < i\}.$$

On peut vérifier que $L(F') = L(F)$. Cette étape peut être appliquée à tout rang k tel que $\text{Elements}(k) = \text{Elements}(k+1)$ pour toute sous-

expression à contraintes d'arité non-minimale de E, et ainsi nous obtenons une expression vérifiant les propriétés du lemme. □

*Démonstration : (**Proposition 4.3**) :* Soit $E^\flat$ l'expression $\emptyset$-linéarisée de E. Remarquons que le langage $L(E^\flat)$ n'est pas modifié, et qu'aucune lettre n'est ajoutée ou éliminée lorsque les opérations suivantes sont appliquées : éliminer les couples inutiles (Définition 4.4), sortir les élements surplombants (Lemme 4.5), transformer $E^\flat$ pour qu'elle soit continue (Lemme 4.6) et d'arité minimale (Lemme 4.7). Il ne nous reste plus qu'à calculer l'expression $E' = \mathrm{h}_E(E^\flat)$. Les preuves constructives de ces lemmes mènent à un algorithme transformant E en une concaténation E' d'ERABT en FoNo telle que $\mathrm{Pos}(E') = \mathrm{Pos}(E)$ et pour tout $k \in \mathrm{Pos}(E)$, $\mathrm{h}_{E'}(k) = \mathrm{h}_E(k)$. □

4.5 Fonctions de Glushkov

Dans cette section, nous montrons comment calculer les fonctions de Glushkov pour les ERABT à partir de la FoNo. Nous présentons tout d'abord comment calculer les fonctions de Glushkov pour les opérateurs de multi-barres, puis pour les opérateurs de multi-tildes. Enfin, nous illustrons ce calcul par un exemple mélangeant les multi-tildes et les multi-barres.

4.5.1 Fonctions de Glushkov pour les Multi-Barres

Le Lemme 4.8 et les deux corollaires qui en découlent permettent d'établir des conditions nécessaires et suffisantes caractérisant l'appartenance de mots particuliers au langage d'une multi-barres.

Lemme 4.8 *Soit $E = \text{——}_B(E_{1,n})$ une ERABT $\emptyset$-linéaire en FoNo. Soit (k, k') un couple de $[\![1, n]\!]^2_{\leq}$ tel que $k < k' - 1$. Considérons un mot $w = w_1 \cdots w_k \cdot w_{k'} \cdots w_n$ tel que pour tout entier j de $[\![1, k]\!] \cup [\![k', n]\!]$, $w_k \in L(\overline{E_k})$. Les deux conditions suivantes sont équivalentes :*

***(1)** Le mot w appartient à $L(E)$,*

***(2)** Pour toute barre (i, f) de B, $[\![i, f]\!] \not\subset [\![k + 1, k' - 1]\!]$.*

Démonstration. Montrons les deux sens de l'équivalence :

(1 ⇒ 2) S'il existe une barre (i, f) de B tel que $k < i \leq f < k'$, alors (i, f) n'est pas vérifiée par w puisque par hypothèse, $w_{k+1} \cdots w_{k'-1} = \varepsilon$, et par conséquent w n'est pas dans $L(E)$.

(1 ⇐ 2) Par hypothèse, toute barre $b' = (i', f')$ de B satisfait une des conditions suivantes : $k' \leq i' < f'$, ou $i' < f' \leq k$, ou $i' \leq k \leq f'$ ou $i' \leq k' \leq f'$. Puisque pour tout j de $[\![1, k]\!] \cup [\![k', n]\!]$, $w_j \neq \varepsilon$, il existe pour chaque barre b' un facteur w_j de w différent de ε vérifiant la barre b'. □

Corollaire 4.3 *Soit $E = \ \_\__B(E_{1,n})$ une ERABT ∅-linéaire en FoNo, k un entier de $[\![1,n]\!]$ et $w = w_k \cdot w_{k+1} \cdots w_n$ un mot tel que pour tout j de $[\![k,n]\!]$, $w_j \in L(\overline{E_j})$. Soit $(1, f_1)$ la barre $\mathrm{Inf}(B)$. Les deux conditions suivantes sont équivalentes :*
(1) *Le mot w est dans $L(E)$,*
(2) *L'entier k est dans $[\![1, f_1]\!]$.*

Corollaire 4.4 *Soit $E = \ \_\__B(E_{1,n})$ une ERABT ∅-linéaire en FoNo, k un entier de $[\![1,n]\!]$ et $w = w_1 \cdot w_2 \cdots w_k$ un mot tel que pour tout j de $[\![1,k]\!]$, $w_j \in L(\overline{E_j})$. Soit (i_m, n) la barre $\mathrm{Sup}(B)$. Les deux conditions suivantes sont équivalentes :*
(1) *Le mot w est dans $L(E)$,*
(2) *L'entier k est dans $[\![i_m, n]\!]$.*

Nous définissons maintenant les fonctions de Glushkov pour les opérateurs de multi-barres, et montrons ensuite qu'elles sont correctes :

Définition 4.9 *Soit $E = \ \_\__B(E_{1,n})$ une ERABT en FoNo avec $B = \big((i_k, f_k)\big)_{k\in[\![1,m]\!]}$, l un entier de $[\![1,n]\!]$, x un élément de $\mathrm{Pos}(E_l)$ et $(i_j, f_j) = \mathrm{Sup}(\mathrm{Barres}(l))$. Les fonctions de Glushkov de la multi-barres E sont définies comme suit :*

$$\textbf{(4.9.1)}\ \mathrm{Pos}(E) = \bigcup_{k=1}^{n} \mathrm{Pos}(E_k) \qquad \textbf{(4.9.2)}\ \mathrm{Null}(E) = \emptyset$$

$$\textbf{(4.9.3)}\ \mathrm{First}(E) = \bigcup_{k=1}^{f_1} \mathrm{First}(E_k) \qquad \textbf{(4.9.4)}\ \mathrm{Last}(E) = \bigcup_{k=i_m}^{n} \mathrm{Last}(E_k)$$

$$\textbf{(4.9.5)}\ \mathrm{Follow}(x,E) = \begin{cases} \mathrm{Follow}(x, E_l) & \text{si } x \notin \mathrm{Last}(E_l) \vee l = n, \\ \mathrm{Follow}(x, E_l) \cup \bigcup_{q=l+1}^{f_{\mathrm{Min}\{j+1,m\}}} \mathrm{First}(E_q) & \text{sinon.} \end{cases}$$

Proposition 4.4 *Soit $E = \ \_\__B(E_{1,n})$ une ERABT en FoNo obtenue par ∅-linéarisation d'une ERABT E' et soit G_E son automate des positions. Alors :*

$$L(E) = L(G_E).$$

Démonstration. La définition des fonctions de Glushkov sur les opérateurs $+$, $\cdot$, et * induisent que :

(1) $x \in \Sigma_E \Leftrightarrow x \in \mathrm{Pos}(E)$,

(2) $\mathrm{Null}(E) = \begin{cases} \{\varepsilon\} & \text{si } \varepsilon \in L(E), \\ \emptyset & \text{sinon,} \end{cases}$

(3) $\exists w = x \cdot w' \in L(E) \Leftrightarrow x \in \mathrm{First}(E)$,

(4) $\exists w = w' \cdot x \in L(E) \Leftrightarrow x \in \mathrm{Last}(E)$,

(5) $\exists w = w' \cdots x \cdot x' \cdots w'' \in L(E) \Leftrightarrow x' \in \mathrm{Follow}(E, x)$.

Montrons alors que les Propositions **(1)** à **(5)** sont toujours vraies lors de l'extension à l'opérateur $\ \_\__B$.

(P1) Pos **:** $x \in \Sigma_E \Leftrightarrow \exists k \in [\![1,n]\!] \mid x \in \Sigma_{E_k} \Leftrightarrow x \in \mathrm{Pos}(E)$.

(P2) Null **:** L'expression E est une ERABT en FoNo, et ainsi $\exists b \in B$. Puisque le mot vide ne vérifie pas la barre b, $\varepsilon \notin L(E)$. D'où $\text{Null}(E) = \emptyset$.

(P3) First **:** ($\Rightarrow$) Puisque E est $\emptyset$-linéaire, $L(E) \neq \emptyset$ et selon (P2) on a $\varepsilon \notin L(E)$. Soit $w = x \cdot w' \in L(E)$ avec $x \in \Sigma_E$. Soit k tel que $x \in \Sigma_{E_k}$ ce qui signifie que $x \in \text{Pos}(E_k)$. Ainsi, il existe $w_k = x \cdot w'_k \in L(E_k)$. Par conséquent, le symbole x est dans $\text{First}(E_k)$ ce qui implique que w peut se décomposer en $w_1 \cdots w_{k-1} \cdot w_k \cdots w_n$ avec $w_1 \cdots w_{k-1} = \varepsilon$ et $w_i \in L(E_i)$ pour $i \geq k$. Puisque $w \in L(E)$, la barre $(1, f_1)$ est vérifiée. Par conséquent, on a $k \in [\![1, f_1]\!]$. Ainsi, selon la Définition (4.9.3), on a $x \in \text{First}(E)$.

($\Leftarrow$) Soit $x \in \text{First}(E)$ et $k \in [\![1, n]\!] \mid x \in \text{First}(E_k)$. Le symbole x est le premier symbole d'au moins un mot $w_k = x \cdot w'$ de $L(E_k)$. De plus, puisque par quotient aucune expression n'est réduite au mot vide et que par $\emptyset$-linéarisation il n'y a plus d'occurrence de l'ensemble vide, pour tout entier j de $[\![1, n]\!]$, il existe w_j dans $L(E_j)$ tel que $w_j \neq \varepsilon$. Considérons le mot $w = w_k \cdot w_{k+1} \cdots w_n$ tel que pour tout entier l de $[\![k, n]\!]$, $w_l \in L(E_l) \setminus \{\varepsilon\}$. Selon le Corollaire 4.3, le mot $w = w_k \cdot w_{k+1} \cdots w_n$ avec $w_k = (x \cdot w')$ est dans $L(E)$ ce qui termine la preuve.

(P4) Last **:** La preuve est similaire à (P3) en raisonnant sur le dernier symbole du mot, et en appliquant le Corollaire 4.4.

(P5) Follow **:** ($\Rightarrow$) Soit $w = w' \cdots x \cdot x' \cdot w''$ un mot de $L(E)$ avec x, x' deux symboles de Σ_E. Soit k l'entier tel que x soit dans $\text{Pos}(E_k)$. Deux cas sont à traiter : **(a)** Si x' est dans $\text{Pos}(E_k)$, alors dans la décomposition w selon $L(E_1) \cdots L(E_n)$, w_k se décompose en $w'_k \cdot x \cdot x' \cdot w''_k$. Ainsi, x' est dans $\text{Follow}(E_k, x)$ et selon la Définition **(4.9.5)**, le symbole x' est dans $\text{Follow}(E, x)$. **(b)** Si x' n'est pas dans $\text{Pos}(E_k)$, alors $w_k = w'_k \cdot x$ et x est dans $\text{Last}(E_k)$. Soit k' l'entier de $[\![k+1, n]\!]$ tel que x' soit dans $\text{Pos}(E_{k'})$. Dans la décomposition de w, on a $w_{k'} = x' \cdot w'_{k'}$. Ainsi, x' est dans $\text{First}(E_{k'})$. Deux cas sont à traiter : **(i)** Si $b_j = \text{Sup}(B)$, alors $f_{\text{Min}(j+1,m)} = f_m = f_j$ et k' est dans $[\![k+1, f_j = n]\!]$. Comme k' est dans $\text{First}(E_{k'})$, selon la Définition **(4.9.5)**, x' est dans $\text{Follow}(E, x)$. **(ii)** Si $b_j \neq \text{Sup}(B)$, considérons $(i_{j'}, f_{j'}) = b_{j'} = \text{Succ}(b_j)$ avec $j' = j + 1 \leq m$. Trivialement, on a $i_{j'} > k$. Dans la décomposition de w, $w_{k+1} \cdots w_{k'-1} = \varepsilon$. Ainsi, puisque la barre $b_{j'}$ est vérifiée par w, cela implique que k' est dans $[\![i_{j'}, f_{j'}]\!]$. En effet, si $k' > f_{j'}$,cela impliquerait que $w_{k+1} \cdots w_{i_{j'}} \cdots w_{f_{j'}} = \varepsilon$ et donc $w \notin L(E)$: contradiction avec l'hypothèse de base. Par conséquent, l'entier k' est dans $[\![k+1, f_{j'} = f_{j+1}]\!]$ et donc $x' \in \text{Follow}(E, x)$.

($\Leftarrow$) Soit x' un symbole de $\text{Follow}(E, x)$. Soit k un entier de $[\![1, n]\!]$ tel que $x \in \text{Pos}(E_k)$. **(a)** Si le symbole x' est dans $\text{Pos}(E_k)$, alors selon la Définition **(4.9.5)**, le symbole x' est dans $\text{Follow}(E_k, x)$ et il existe un mot $w_k = w'_k \cdot x \cdot x' \cdot w''_k$ dans $L(E_k)$. De plus, puisque par quotient aucune expression n'est réduite au mot vide et que par $\emptyset$-linéarisation il n'y a plus d'occurrence de l'ensemble vide, pour tout j de $[\![1, n]\!]$, il existe un mot w_j de $L(E_j)$ tel que w_j soit différent de ε. Soit $w = w_1 \cdots w_k \cdots w_n$ un mot

tel que pour tout j de $[\![1,n]\!]$, $w_j \neq \varepsilon$. Par conséquent, toute barre de B est vérifiée et $w \in L(E)$. **(b)** Si $x' \notin \mathrm{Pos}(E_k)$, soit $k' \in [\![1,n]\!] \mid x' \in \mathrm{Pos}(E_k)$. Dans ce cas, selon la Définition **(4.9.5)**, x' est nécessairement dans $\mathrm{First}(E_{k'})$ et x est dans $\mathrm{Last}(E_k)$. Ainsi il existe deux mots w_k et $w_{k'}$ tels que $w_k = w'_k \cdot x$ et $w_{k'} = x' \cdot w''_{k'}$. Considérons le mot $w = w_1 \cdots w_k \cdot w_{k'} \cdots w_n$ tel que pour tout i de $[\![1,n]\!]$, w_i est dans $L(E_i) \setminus \{\varepsilon\}$. Si $b_j = \mathrm{Sup}(\mathrm{Barres}(k))$, on sait que $k' \in [\![k+1, f_{\mathrm{Min}(j+1,m)}]\!]$. Ainsi, $\nexists (i,f) \in B \mid k < i < f < k'$ et donc, selon le Lemme 4.8, $w \in L(E)$. □

4.5.2 Fonctions de Glushkov pour les Multi-Tildes

Nous définissons maintenant les fonctions de Glushkov pour les opérateurs de multi-tildes, et montrons ensuite qu'elles sont correctes :

Définition 4.10 *Soit $E = \sim_T(E_{1,n})$ une ERABT $\emptyset$-linéaire en FoNo avec $T = \big((i_k, f_k)\big)_{k\in[\![1,m]\!]}$, l un entier de $[\![1,n]\!]$ et x un élément de $\mathrm{Pos}(E_l)$. Les fonctions de Glushkov de la multi-tildes E sont définies comme suit :*

(4.10.1) $\mathrm{Pos}(E) = \bigcup_{k=1}^{n} \mathrm{Pos}(E_k)$ ***(4.10.2)*** $\mathrm{Null}(E) = \begin{cases} \varepsilon & \text{si } E \text{ est } (1,n)\text{-traversable,} \\ \emptyset & \text{sinon.} \end{cases}$

(4.10.3) $\mathrm{First}(E) = \begin{cases} \mathrm{First}(E_1) \cup \\ \{\mathrm{First}(E_k) \mid k \in [\![2,n]\!] \wedge E \text{ est } (1,k-1)\text{-traversable}\} \end{cases}$

(4.10.4) $\mathrm{Last}(E) = \begin{cases} \mathrm{Last}(E_n) \cup \\ \{\mathrm{Last}(E_k) \mid k \in [\![1,n-1]\!] \wedge E \text{ est } (k+1,n)\text{-traversable}\} \end{cases}$

(4.10.5) $\mathrm{Follow}(x,E) = \begin{cases} \mathrm{Follow}(x,E_l) & \text{si } x \notin \mathrm{Last}(E_l) \\ & \vee\ l = n, \\ \mathrm{Follow}(x,E_l) \cup \mathrm{First}(x,E_{l+1}) \cup & \\ \{\mathrm{First}(E_{l'}) \mid l' \in [\![l+2,n]\!] \wedge & \\ E \text{ est } (l+1,l'-1)\text{-traversable}\} & \text{sinon.} \end{cases}$

Proposition 4.5 *Soit $E = \sim_T(E_{1,n})$ une ERABT $\emptyset$-linéaire en FoNo obtenue par $\emptyset$-linéarisation d'une ERABT E', et soit G_E son automate de Glushkov. Alors :*

$$L(E) = L(G_E).$$

Démonstration. La définition des fonctions de Glushkov sur les opérateurs $+$, $\cdot$ et * induisent :

(P1) $x \in \Sigma_E \Leftrightarrow x \in \mathrm{Pos}(E)$,

(P2) $\mathrm{Null}(E) = \begin{cases} \{\varepsilon\} & \text{si } \varepsilon \in L(E), \\ \emptyset & \text{sinon,} \end{cases}$

(P3) $\exists w = x \cdot w' \in L(E) \Leftrightarrow x \in \mathrm{First}(E)$,

(P4) $\exists w = w' \cdot x \in L(E) \Leftrightarrow x \in \mathrm{Last}(E)$,

(P5) $\exists w = w' \cdots x \cdot x' \cdots w'' \in L(E) \Leftrightarrow x' \in \mathrm{Follow}(x,E)$.

Montrons que les propositions **(P1)** à **(P5)** sont toujours vraies lors de l'extension aux opérateurs de multi-tildes.

(P1) Pos : $x \in \Sigma_E \Leftrightarrow \exists k \in [\![1,n]\!] \mid x \in \Sigma_{E_k} \Leftrightarrow x \in \mathrm{Pos}(E)$.

(P2) Null **:** Selon le Corollaire 4.2, $\varepsilon \in L(E) \Leftrightarrow E$ est $(1,n)$-traversable. Ainsi, l'extension de Null(E) est correcte.
(P3) First **:** ($\Rightarrow$) Puisque E est $\emptyset$-linéaire, on a $L(E) \neq \emptyset$ et $L(E) \neq \varepsilon$. Soit $w = x \cdot w' \in L(E)$ avec $x \in \Sigma_E$. Soit k tel que $x \in \Sigma_{E_k}$, ce qui signifie que $x \in \text{Pos}(E_k)$. Ainsi, il existe $x \cdot w'_k \in L(E_k)$. Par conséquent, $x \in \text{First}(E_k)$. Si $k = 1$, selon la Définition 4.10.3, $x \in \text{First}(E)$. Sinon, cela implique que $w = w_1 \cdots w_{k-1} \cdot w_k \cdot w_{k+1} \cdots w_n$ avec $w_k = x \cdot w'_k$ et $w_1 \cdots w_{k-1} = \varepsilon$, ainsi $w_1 \cdots w_k \neq \varepsilon$. Selon la Proposition 4.1 et puisque $w_1 \cdots w_{k-1}$ est ε-maximal, il existe une liste libre S contenant le couple $(1, k-1)$ tel que E est S-traversable. Ainsi, E est $(1, k-1)$-traversable et selon la Définition 4.10.3, $x \in \text{First}(E)$.

($\Leftarrow$) Supposons que $x \in \text{First}(E)$ et soit $k \in [\![1, n]\!]$ tel que $x \in \text{First}(E_k)$. Puisque E est $\emptyset$-linéaire $\forall i \in [\![1, n]\!]$, $L(E_i) \neq \emptyset$ et $L(E_i) \neq \{\varepsilon\}$. Supposons que $w = w_k \cdots w_n$ avec $\forall i \in [\![k, n]\!], w_i \in L(E_i)$ et $w_i \neq \varepsilon$. Si $k = 1$, le mot w est dans $L(E_{1\cdots n})$ et selon la Définition 3.4, $L(E_{1\cdots n}) \subset L(E)$ et ainsi $w \in L(E)$. Si $k \neq 1$, alors, selon la Définition 4.10.3, E est $(1, k-1)$-traversable, et selon le Corollaire 4.2, $w \in L(E)$.
(P4) Last **:** La preuve est similaire à (P3) en raisonnant sur le dernier symbole des mots.
(P5) Follow **:** Soit k l'entier de $[\![1, n]\!]$ tel que $x \in \text{Pos}(E_k)$.

($\Rightarrow$) Soit x' un élément de Follow(x, E) et k' l'entier de $[\![1, n]\!]$ tel que $x' \in \text{Pos}(E_{k'})$. **(1)** Supposons que $k = k'$. Ainsi il existe $w_k = w'_k \cdot x \cdot x' \cdot w''_k$ dans $L(E_k)$. Soit $w = w_1 \cdots w_k \cdots w_n$ un mot tel que pour tout entier i de $[\![1, n]\!] \setminus \{k\}$, w_i est dans $L(E_i) \setminus \{\varepsilon\}$ (cette décomposition existe puisque l'expression est $\emptyset$-linéaire). Selon la Définition 3.4, $w \in L(E_{1\cdots n}) \subset L(E)$, et donc $w \in L(E)$. **(2)** Supposons $k \neq k'$. Selon la Définition 4.10.5, x est dans Last(E_k), $k \neq n$ et x' est dans First$(E_{k'})$. Ainsi, il existe $w_k = w'_k \cdot x$ dans $L(E_k)$ et $w_{k'} = x' \cdot w''_{k'}$ dans $L(E_{k'})$. Soit $w = w_1 \cdots w_k \cdots w_n$ un mot tel que pour tout entier i de $[\![1, n]\!] \setminus \{k\}$, w_i est dans $L(E_i) \setminus \{\varepsilon\}$ (cette décomposition existe puisque l'expression est $\emptyset$-linéaire). **(a)** Supposons $k' = k + 1$. Selon la Définition 3.4, w est dans $L(E_1 \cdots E_k \cdot E_{k+1} \cdots E_n) \subset L(E)$, et $w \in L(E)$. **(b)** Supposons que $k' > k + 1$. Si x' appartient à Follow(x, E), alors selon la Définition 4.10.5 , E est $(k+1, k'-1)$-traversable, et selon le Corollaire 4.2, $w \in L(E)$.

($\Leftarrow$) Supposons qu'il existe un mot $w = w' \cdot x \cdot x' \cdot w''$ dans $L(E)$. Il existe $k' \mid n > k' > k > 1$ tel que x' est dans Pos$(E_{k'})$. **(1)** Supposons $k = k'$. Puisque l'expression E est $\emptyset$-linéaire, w admet une unique décomposition selon les facteurs E_i et w est égal à $w_1 \cdots w_k \cdots w_n$ avec pour tout entier i de $[\![1, n]\!]$, w_i dans $L(E_i)$ et $w_k = w'_k \cdot x \cdot x' \cdot w''_k$. Ainsi, x' est dans Follow(x, E_k) et donc x' appartient à Follow(x, E). **(2)** Supposons $k \neq k'$. Puisque l'expression E est $\emptyset$-linéaire, w admet une décomposition unique sur les facteurs E_i et $w = w_1 \cdots w_k \cdot w_{k'} \cdots w_n$ telle que pour tout entier i de $[\![1, n]\!]$, w_i est dans $L(E_i)$, $w_k = w'_k \cdot x$ et $w_{k'} = x' \cdot w''_{k'}$. **(a)** Si $k' = k + 1$, comme x' appartient à First(E_{k+1}), x' est dans Follow(x, E). **(b)** Si $k' > k + 1$, selon la

Proposition 4.1, il existe une liste libre $S = \left((i_j, f_j)\right)_{j\in[\![1,\#S]\!]}$ telle que pour tout (i, f) de S, le facteur $w_i \cdots w_f$ est ε-maximal, et E est S-traversable. Comme $w_{k+1} \cdots w_{k'-1}$ est ε-maximal, $(k+1, k'-1)$ appartient à S et ainsi E est $(k+1, k'-1)$-traversable. Par conséquent, x' est dans Follow(x, E) selon la Définition 4.10.5. □

4.5.3 Exemple de Construction

Soit $E = \overline{\overline{(E_1 + \varepsilon)(E_2)}\ (E_3)}$ une ERABT en FoNo définie par les expressions $E_1 = \widetilde{(a\ b\ c\ d\ e)}$, $E_2 = (f + \varepsilon)$ et $E_3 = (g + \varepsilon)$. Soit $E' = \overline{\overline{(E'_1 + \varepsilon)(E'_2)}\ (E'_3)}$ l'expression $\emptyset$-linéarisée de E. Les expressions E'_1, E'_2 et E'_3 sont alors $E'_1 = \widetilde{(1\ 2\ 3\ 4\ 5)}$, $E'_2 = (6 + \varepsilon)$ et $E'_3 = (7 + \varepsilon)$.

Considérons l'expression E'_1. Puisque la première tilde couvre les expressions 1 et 2, l'expression E'_1 est $(1, 2)$-traversable et par conséquent la fonction First(E'_1) est l'union des First$(1) = \{1\}$ et First$(3) = \{3\}$. Ainsi First$(E'_1) = \{1, 3\}$. L'ensemble Follow$(1, E'_1)$ est l'union des ensembles Follow$(1, 1) = \emptyset$, First$(2) = \{2\}$ puisque la position 1 appartient à l'ensemble Last$(1) = \{1\}$ et First$(5) = \{5\}$ puisque l'expression E'_1 est $(2, 4)$-traversable. Ainsi Follow$(1, E'_1) = \{2, 5\}$.

Considérons l'expression E'. La fonction First(E') contient les First des expressions sous la première barre, soit les ensembles First$(E'_1) = \{1, 3\}$ et First$(E'_2) = \{6\}$. La fonction Last(E') contient les Last des expressions sous la dernière barre, soit Last$(E'_2) = \{6\}$ et Last$(E'_3) = \{7\}$. Remarquons que la position 3, appartenant à Last(E'_1), n'appartient pas à Last(E'), puisque la barre $(2, 3)$ est présente.

En appliquant le calcul de toutes les fonctions de Glushkov pour E à partir de E', nous obtenons alors les résultats de la Table 4.1 menant alors par délinéarisation à l'automate des positions de la Figure 4.2.

Pos(E)	$\{1, 2, 3, 4, 5, 6, 7\}$
Null(E)	$\emptyset$
First(E)	$\{1, 3, 6\}$
Last(E)	$\{6, 7\}$

Follow$(1, E)$	$\{2, 5\}$
Follow$(2, E)$	$\{3\}$
Follow$(3, E)$	$\{4, 6, 7\}$
Follow$(4, E)$	$\{5\}$
Follow$(5, E)$	$\{6, 7\}$
Follow$(6, E)$	$\{7\}$
Follow$(7, E)$	$\emptyset$

TABLE 4.1 – *Les Fonctions de Glushkov pour l'Expression E.*

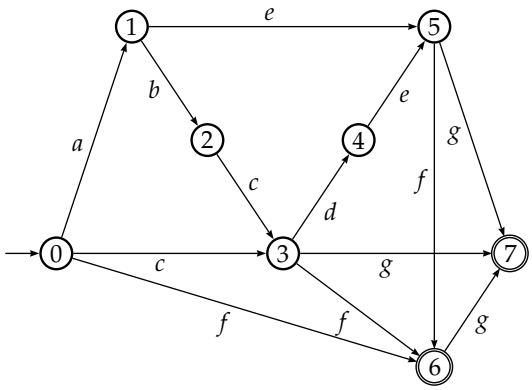


FIGURE 4.2 – *L'Automate de Glushkov de l'Expression E.*

4.6 Conclusion

Dans ce chapitre, nous avons établi une forme particulière pour les ERABT, la forme normalisée, nous permettant de distinguer les langages engendrés par l'application de deux listes de barres ou de tildes distinctes sur une même liste d'expressions. La notion clé est celle de l'utilité, qui tient compte de l'effet de l'action d'un couple lors de l'application d'un opérateur à contraintes.

La forme normalisée d'une ERABT nous a permis d'établir sous une forme moins complexe la construction d'un automate fini reconnaissant le langage qu'elle dénote, et cela en étendant les fonctions de Glushkov. Le nombre d'automates à $(n+1)$ états isomorphes à l'automate des positions d'une expressions à n lettres se voit alors augmenté.

Nous montrerons dans le chapitre suivant que le pouvoir de factorisation de l'utilisation simultanée des multi-tildes et des multi-barres peut se voir augmenter en utilisant à la fois des listes de barres et des listes de tildes appliquées sur une même liste d'expressions. Par conséquent, nous étudierons la conversion d'un automate en une expression à contraintes à travers ces nouvelles expressions, les multi-tildes-barres.

Chapitre 5

Les Multi-Tildes-Barres

La totalité est plus que la somme des parties.
Aristote

L'union fait la force.
Esope

Oui. Mais la force de qui ?
Emile Chartier, dit Alain

Sommaire

Dans ce chapitre, nous présentons une nouvelle famille d'opérateurs, les multi-tildes-barres, permettant d'étendre la notion de multi-barres et de multi-tildes en autorisant la combinaison de tildes et de barres dans un même opérateur. La Section 5.1 présente l'intérêt à disposer d'un opérateur de multi-tildes-barres en exhibant un exemple dans lequel l'utilisation des multi-tildes et des multi-barres peut être étendue. En caractérisant les mots contenus dans le langage dénoté, la Section 5.2 définit les opérateurs de multi-tildes-barres, dont la rationalité ainsi que la compatibilité avec la ∅-linéarisation sont montrées dans la Section 5.3. Enfin, une forme particulière est définie dans la Section 5.4, permettant de résoudre les collisions de langages.

5.1 Introduction

Les opérateurs à contraintes, constitués de la famille des multi-barres et de celle des multi-tildes, permettent une factorisation plus importante que les opérateurs rationnels simples. Un opérateur à contraintes permet d'appliquer simultanément un ensemble de contraintes du même type (des barres pour une multi-barres, des tildes pour une multi-tildes), et ne convient pas s'il s'agit d'appliquer simultanément des contraintes de types différents. L'exemple suivant le montre.

Soit k un entier supérieur ou égal à 3. Considérons le langage L_k reconnu par le NFA à $(k+1)$ états $A_k = (\Sigma_k, Q_k, I_k, F_k, \delta_k)$ avec

- $\Sigma_k = \{a_1, \ldots, a_k\}$,
- $Q_k = [\![0, k]\!]$,
- $I_k = \{0\}$,
- $F_k = \{k-2, k\}$,
- $\delta_k = \{(i, a_f, f) \in Q_k \times \Sigma_k \times Q_k \mid f = i+1 \ \vee \ f = i+3\}$,

représenté Figure 5.1. Remarquons que le langage L_k est tel que :

$$L_k = \left\{ \begin{array}{l} a_{k_1} \cdots a_{k_m} \in \Sigma_k^* \mid \forall i \in [\![1, m-1]\!], \ k_{i+1} = k_i + 1 \ \vee \ k_{i+1} = k_i + 3 \\ \qquad\qquad\qquad \wedge \ k_1 \in \{1, 3\} \\ \qquad\qquad\qquad \wedge \ k_m \in \{n-2, n\} \end{array} \right\}.$$

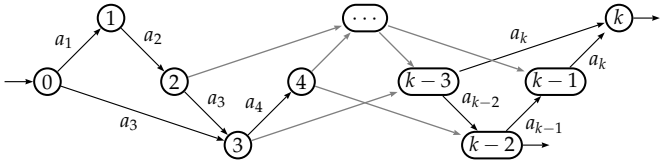


FIGURE 5.1 – *Un NFA à* $(k+1)$ *États Reconnaissant* L_k.

Moins formellement, le langage L_k correspond à l'ensemble des mots commençant par la lettre a_1 ou par la lettre a_3, se terminant par la lettre a_{k-2} ou par la lettre a_k, et telle que toute lettre a_i apparaissant dans ce mot est suivie soit par a_{i+1}, soit par a_{i+3}. Ainsi :

$$\begin{aligned} L_3 &= \{a_1a_2a_3, a_1, a_3\}, \\ L_4 &= \{a_1a_2a_3a_4, a_1a_4, a_1a_2, a_3a_4\}, \\ L_5 &= \{a_1a_2a_3a_4a_5, a_1a_4a_5, a_1a_2a_3, a_1a_2a_5, a_3, a_3a_4a_5\}. \end{aligned}$$

En fait, chacun de ces langages correspond à l'action de tildes et de barres, appliquées simultanément sur une liste d'expressions. En effet, remarquons que les langages L_3, L_4 et L_5 sont respectivement égaux aux ensembles suivants :

$$\begin{aligned} L_3 &= L(a_1a_2a_3 + a_1 + a_3) \\ &= L(a_1a_2a_3 + a_1 \widetilde{a_2a_3} + \widetilde{a_1a_2}\, a_3) \\ &= L(\widetilde{a_1 \quad \widetilde{a_2} \quad a_3}) \\ &= L(\sim_{(1,2),(2,3)}(a_1, a_2, a_3)), \end{aligned}$$

$$
\begin{aligned}
L_4 &= L(a_1a_2a_3a_4 + a_1a_4 + a_1a_2 + a_3a_4)\\
&= L(a_1a_2a_3a_4 + a_1\,\widetilde{a_2a_3}\,a_4 + a_1a_2\,\widetilde{a_3a_4} + \widetilde{a_1a_2}\,a_3a_4)\\
&= L(\overline{\widetilde{a_1 \quad a_2} \quad \widetilde{a_3 \quad a_4}})\\
&= L(\overline{\sim_{(1,2),(2,3),(3,4)}(a_1,a_2,a_3,a_4)}),\\
&= L(\,\text{—}_{(1,1)}(\sim_{(1,2),(2,3),(3,4)}(a_1,a_2,a_3,a_4))),\\
L_5 &= L(a_1a_2a_3a_4a_5 + a_1a_4a_5 + a_1a_2a_3 + a_1a_2a_5 + a_3a_4a_5 + a_3)\\
&= L(a_1a_2a_3a_4a_5 + a_1a_2\,\widetilde{a_3a_4}\,a_5 + \widetilde{a_1a_2}\,a_3a_4a_5\\
&\quad + a_1\,\widetilde{a_2a_3}\,a_4a_5 + a_1a_2a_3\,\widetilde{a_4a_5} + \widetilde{a_1a_2}\,a_3\,\widetilde{a_4a_5})\\
&= L((\overline{\widetilde{a_1 \quad a_2} \quad \widetilde{a_3 \quad a_4}}) \cdot a_5\\
&\quad + a_1 \cdot (\overline{\widetilde{a_2 \quad a_3} \quad \widetilde{a_4 \quad a_5}})\\
&\quad + \widetilde{a_1a_2}\,a_3\,\widetilde{a_4a_5}).
\end{aligned}
$$

Les langages L_3 et L_4 sont dénotés par des ERABT respectivement de largeur 3 et 4. Cependant, le langage L_5 ne peut être dénoté par une ERABT de largeur 5. En effet, la tilde $(1,2)$ (resp. $(3,4)$) est nécessaire puisque le mot $a_3a_4a_5$ (resp. $a_1a_2a_5$) appartient à L_5 ; la barre $(1,4)$ est nécessaire puisque le mot a_5 n'appartient pas à L_5. De même, la tilde $(2,3)$ (resp. $(4,5)$) est nécessaire puisque le mot $a_1a_4a_5$ (resp. $a_1a_2a_3$) appartient à L_5 ; la barre $(2,5)$ est nécessaire puisque le mot a_1 n'appartient pas à L_5. Par conséquent, il faut considérer l'action simultanée des tildes de $T = \{(1,2),(2,3),(3,4),(4,5)\}$ et des barres de $B = \{(1,4),(2,5)\}$ qui se chevauchent mutuellement.

Dans la section suivante, nous définissons de nouveaux opérateurs pouvant décrire ce type de situation, les *multi-tildes-barres*. Ainsi, le langage L_5 est dénoté par l'expression en multi-tildes-barres suivante :

$$L_5 = L(\overline{\widetilde{a_1 \quad a_2} \quad \widetilde{a_3 \quad a_4} \quad a_5}).$$

5.2 Les Opérateurs de Multi-Tildes-Barres

Un opérateur de multi-tildes-barres (MTB) est paramétré par deux listes de couples, T une liste de tildes et B une liste de barres, toutes deux disjointes. Son application sur une liste d'expressions $E_{1,n}$ fournit une expression dont le langage contient les mots générés par une sous-liste libre de T, mais cette génération est restreinte par l'action de B.

Exemple 5.1 *Considérons l'expression* $E = \overline{\sim}_{T;B}(1, 2+\varepsilon, 3+\varepsilon, 4)$ *avec* $T = \{(1,2),(3,4)\}$ *et* $B = \{(2,3)\}$ *représentée graphiquement par :*

$$E = \overline{\widetilde{(1)} \;\; \widetilde{(2+\varepsilon)}} \qquad \overline{\widetilde{(3+\varepsilon)} \;\; \widetilde{(4)}}\,.$$

Considérons le mot $1 \cdot \varepsilon \cdot \varepsilon \cdot 4$. *Ce mot est généré par la liste de tildes vide (qui est une sous-liste libre de T). Cependant, la présence de la barre* $(2,3)$ *exclut ce mot*

du langage. Considérons le mot vide ε. Remarquons que ce mot est généré par la liste libre de tildes $((1,2),(3,4))$. Bien que le mot vide se décompose en un mot $w_1 \cdot w_2 \cdot w_3 \cdot w_4$ tel que pour tout k de $[\![1,4]\!]$, $w_k = \varepsilon$ et qu'ainsi $w_2 \cdot w_3 = \varepsilon$, nous ne souhaitons pas que la barre retire ce mot du langage. En fait, l'action de la barre (i,f) est d'interdire tout ε-facteur généré sous l'intervalle $[\![i,f]\!]$, et uniquement sous cet intervalle. Ici, le facteur $w_2 \cdot w_3$ est en fait généré à partir d'un intervalle plus grand, utilisant les tildes $(1,2)$ et $(3,4)$. Ainsi, pour interdire le ε-facteur $w_1 w_4$, la barre $(1,4)$ aurait été nécessaire. Par conséquent, le mot vide ε appartient au langage dénoté par E.

Cette restriction de l'action des tildes par une barre est en fait une extension de la notion de vérification de barre par un mot, définie comme suit :

Définition 5.1 *Soit $w = w_1 \cdots w_n$ un mot généré par une liste libre T de $\mathcal{S}_n$ à partir d'une liste d'expressions $E_{1,n}$. Soit $b = (i,f)$ un couple de $[\![1,n]\!]^2_{\leq} \setminus T$. Nous dirons que la barre b est* vérifiée par w selon T *si au moins une des trois propositions suivantes est vérifiée :*

(1) *il existe un couple t dans T tel que t chevauche b,*

(2) *il existe un couple t dans T tel que b soit inclus dans t,*

(3) *$w_i \cdots w_f \neq \varepsilon$.*

On définit alors l'action d'un opérateur de multi-tildes-barres.

Définition 5.2 *Soit n un entier positif. Soit $E_{1,n}$ une liste d'expressions sur un alphabet Σ. Soit B et T deux listes de $\mathcal{S}_n$ telles que B et T soient disjointes. La multi-tildes-barres $E = \overline{\sim}_{T;B}(E_{1,n})$ dénote le langage :*

$$L(\overline{\sim}_{T;B}(E_{1,n})) = \left\{ \begin{array}{c} w \in \Sigma^* \mid \text{il existe une sous-liste libre } T' \text{ de } T \\ \text{telle que } w \text{ est généré par } T' \\ \text{et telle que } w \text{ vérifie toute barre de } B \text{ selon } T' \end{array} \right\}.$$

À partir de maintenant, toute expression utilisant un opŕateur de multi-barres, de multi-tildes ou de multi-tildes-barres sera appelée *expression à contraintes*.

Si une des expressions de la liste sur laquelle s'applique un opérateur de multi-tildes-barres est réduite au mot vide, nous appliquerons un quotient quasi-similaire à celui sur les ERABT. La seule différence entre ces deux quotients correspond aux cas suivants :

Exemple 5.2 *Soit $E_1 = \overline{\sim}_{(1,3);(2,3)}(\varepsilon, a+\varepsilon, b+\varepsilon)$ et $E_2 = \overline{\sim}_{(2,3);(1,3)}(\varepsilon, a+\varepsilon, b)$ deux multi-tildes-barres représentées graphiquement comme suit :*

$$E_1 = \overbrace{\varepsilon\ \overline{(a+\varepsilon)(b+\varepsilon)}}\ ,\ E_2 = \overline{\varepsilon\ \overbrace{(a+\varepsilon)(b)}}\ .$$

Remarquons que l'expression E_1 (resp. E_2) est équivalente à $\overbrace{(a+\varepsilon)(b+\varepsilon)}$ (resp. $\overline{(a+\varepsilon)(b)}$) et que la barre (resp. la tilde) $(2,3)$ peut être éliminée.

Le quotient se définit formellement comme suit. Soit n un entier supérieur à 0. Soit $E_{1,n}$ une liste d'expressions, soit B et T deux listes disjointes de $\mathcal{S}_n$ et soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une multi-tildes-barres.

Supposons qu'il existe un entier k de $[\![1,n]\!]$ tel que $E_k = \varepsilon$.
(I) Si $n = 1$, on a $E_1 = \varepsilon$. Si $B = \emptyset$, alors $E \sim \varepsilon$. Sinon $B = ((1,1))$ et $E \sim \emptyset$.
(II) Supposons $n \geq 2$. Soit $E' = \overline{\sim}_{T';B'}(E'_{1,n})$ avec :

$$E'_{k'} = \begin{cases} E'_{k'} = E_{k'} & \text{si } k' \in [\![1,k-1]\!], \\ E'_{k'} = E_{k'+1} & \text{si } k' \in [\![k,n-1]\!], \end{cases}$$
$$\begin{aligned}
T'_1 &= \{(i,f) \in T \mid f < k\}, \\
T'_2 &= \{(i-1,f-1) \mid (i,f) \in T \ \wedge \ i > k+1\} \\
&\quad \cup \{(i-1,f-1) \mid (i,f) \in T \ \wedge \ i = k+1 \wedge (k,f) \notin T \cup B\}, \\
T'_3 &= \{(i,f-1) \mid (i,f) \in T \ \wedge \ i \leq k \ \wedge \ k \leq f\}, \\
T' &= T'_1 \cup T'_2 \cup T'_3. \\
B'_1 &= \{(i,f) \in B \mid f < k\}, \\
B'_2 &= \{(i-1,f-1) \mid (i,f) \in B \ \wedge \ i > k+1\} \\
&\quad \cup \{(i-1,f-1) \mid (i,f) \in B \ \wedge \ i = k+1 \wedge (k,f) \notin T \cup B\}, \\
B'_3 &= \{(i,f-1) \mid (i,f) \in B \ \wedge \ i \leq k \ \wedge \ k \leq f\}, \\
B' &= B'_1 \cup B'_2 \cup B'_3.
\end{aligned}$$

Il est facile de vérifier que $E \sim E'$. Nous considérons donc pour la suite que toute multi-tildes-barres $E = \overline{\sim}_{T;B}(E_{1,n})$ avec $n > 0$ est telle que pour tout k de $[\![1,n]\!]$, $L(E_k) \neq \{\varepsilon\}$.

Les opérateurs de multi-tildes-barres possèdent un pouvoir de représentation au moins aussi fort que les opérateurs de multi-tildes et de multi-barres :

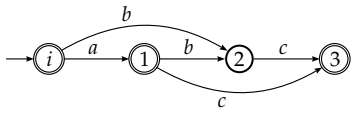
Lemme 5.1 *Soit n un entier positif. Soit $E_{1,n}$ une liste d'expressions sur un alphabet Σ. Soit B et T deux listes disjointes de $\mathcal{S}_n$. Alors :*
(1) *Si $T = \emptyset$, alors $L(\overline{\sim}_{T;B}(E_{1,n})) = L(\overline{\ }_B(E_{1,n}))$,*
(2) *Si $B = \emptyset$, alors $L(\overline{\sim}_{T;B}(E_{1,n})) = L(\sim_T(E_{1,n}))$.*

Démonstration. **(1)** Supposons que T est vide. Remarquons alors que la vérification de barre selon une sous-liste de T (Définition 5.1) est équivalente à la vérification de barre (Définition 3.1), puisque si un mot w vérifie une barre (i,f) selon la liste vide $\emptyset$, alors il vérifie la barre (i,f). Ainsi la Définition 5.2 est équivalente à la Définition 3.2.

(2) Si $B = \emptyset$, puisque qu'aucune barre n'est à vérifier pour qu'un mot appartienne au langage, la Définition 5.2 est équivalente à la Définition 3.4, puisque tout mot généré par une liste libre T vérifie toujours toutes les barres de la liste vide. □

Les opérateurs de multi-tildes-barres permettent d'augmenter le nombre de langages reconnus par un automate à $(n+1)$ états qui n'est pas de Glushkov mais de Glushkov étendu, et dénotés par une expression rationnelle étendue avec n occurrences de symboles. Par exemple, consi-

dérons l'automate de la Figure 5.2. L'expression rationnelle simple E', de largeur 5 est une expression simple minimale. L'ERABT E'' est de largeur 4. L'expression E, utilisant un opérateur de multi-tildes-barres est de largeur 3. Les méthodes de conversion entre les NFA et les expressions en multi-tildes-barres seront présentées dans le chapitre suivant.



$$\begin{aligned} E' &= \underline{(a(b+\varepsilon)+b)c+a+\varepsilon} \\ E'' &= \underbrace{(\widetilde{a})(\widetilde{b})}c+\widetilde{a} \\ E &= \overline{\widetilde{a}\,\widetilde{b}}\;c \\ &= \overline{\widetilde{\ }}_{(1,1),(2,2),(2,3);(1,2)}(a,b,c) \end{aligned}$$

FIGURE 5.2 – *Un Exemple de Factorisation par un Opérateur de Multi-Tildes-Barres.*

5.3 Rationnels et Compatibles avec la ∅-Linéarisation

Nous montrons dans cette section que les opérateurs de multi-tildes-barres sont des opérateurs rationnels et compatibles avec la ∅-linéarisation. Nous commençons par montrer la compatibilité avec la ∅-linéarisation, puis nous montrons que le langage dénoté par une expression ∅-linéaire en multi-tildes-barres est rationnel.

5.3.1 Compatibilité avec la ∅-Linéarisation

Les opérateurs de multi-barres et de multi-tildes sont compatibles avec la ∅-linéarisation. Les multi-tildes-barres, correspondant à la combinaison de ces deux notions, sont eux aussi compatibles avec la ∅-linéarisation.

Lemme 5.2 *Soit $\mathcal{O}$ l'ensemble des opérateurs défini par :*

$$\mathcal{O} = \{+,\cdot,^*\} \cup \bigcup_{T,B\mid \exists n\in\mathbb{N},(T,B\in\mathcal{S}_n)\wedge(T\cap B=\emptyset)} \overline{\widetilde{\ }}_{T;B}.$$

Les opérateurs de $\mathcal{O}$ sont compatibles avec la linéarisation.

Démonstration. Soit E une expression sur les opérateurs de $\mathcal{O}$, et soit $E^\flat$ son expression ∅-linéarisée. Nous étendons l'induction sur la largeur de E et sur son nombre d'opérateurs présentée dans la preuve du Lemme 1.9. Posons $E^\flat = \overline{\widetilde{\ }}_{T;B}(E'_{1,n})$.

Soit w un mot de $L(E)$. Selon la Définition 5.2, il existe une sous-liste libre T' de T telle que T' génère w et telle que w vérifie toute barre de B selon T'. Ainsi, selon la Définition 3.3, w se décompose en $w_1\cdots w_n$ avec pour tout k de $[\![1,n]\!]$, $w_k = \varepsilon$ s'il existe (i,f) dans T' tel que $k \in [\![i,f]\!]$, $w_k \in L(E_k)$ sinon. Par hypothèse d'induction, pour tout k de $[\![1,n]\!]$, il existe un mot w'_k dans $\mathrm{Pos}(E)^*$ tel que $w'_k = \varepsilon$ s'il existe (i,f) dans T' tel que $k \in [\![i,f]\!]$, $w'_k \in L(E'_k)$ sinon. Ainsi, T' génère $w' = w'_1\cdots w'_n$ et w'

vérifie toutes les barres de B selon T' (sinon contradiction avec w vérifie toutes les barres de B selon T'). Selon la Définition 5.2, w' est dans $L(E^\flat)$ et puisque par construction $\mathrm{h}_E(w') = w$, w est dans $\mathrm{h}_E(L(E^\flat) \cap \mathrm{Pos}(E)^*)$.

Soit w un mot de $\mathrm{h}_E(L(E^\flat) \cap \mathrm{Pos}(E)^*)$. Alors il existe w' dans $L(E^\flat)$ tel que $\mathrm{h}_E(w') = w$. Selon la Définition 5.2, il existe une sous-liste libre T' de T telle que T' génère w et telle que w' vérifie toute barre de B selon T'. Ainsi, selon la Définition 3.3, w' se décompose en $w'_1 \cdots w'_n$ avec pour tout k de $[\![1, n]\!]$, $w'_k = \varepsilon$ s'il existe (i, f) dans T' tel que $k \in [\![i, f]\!]$, $w'_k \in L(E'_k)$ sinon. Considérons le mot $w_1 \cdots w_n$ tel que $w_k = \mathrm{h}(w'_k)$ pour tout k de $[\![1, n]\!]$. Par construction, $w = w_1 \cdots w_n$. Par hypothèse d'induction, les langages $L(E_k)$ et $\mathrm{h}_E(L(E'_k) \cap \mathrm{Pos}(E)^*)$ sont égaux. Par conséquent, pour tout k de $[\![1, n]\!]$, il existe un mot w_k dans Σ^* tel que $w_k = \varepsilon$ s'il existe (i, f) dans T' tel que $k \in [\![i, f]\!]$, $w_k \in L(E_k)$ sinon. Ainsi, T' génère $w = w_1 \cdots w_n$ et w vérifie toutes les barres de B selon T' (sinon contradiction avec w' vérifie toutes les barres de B selon T'). Finalement, selon la Définition 5.2, $w \in L(E)$. □

5.3.2 Rationalité du Langage d'une MTB ∅-Linéaire

Nous détaillons ici le calcul du langage d'une expression en multi-tildes-barres ∅-linéaire en séparant quatre cas, du plus simple au plus complexe. Ces formules utilisent des opérations rationnelles, et nous en déduisons donc la rationalité du langage dénoté par une expression en multi-tildes-barres quelconque.

Cas d'une Barre Unique ne Chevauchant pas de Tildes :

Il s'agit du cas le plus simple, où chacun des opérateurs élémentaires (barres ou tildes) peut être directement remplacé dans l'expression par une barre simple ou par une tilde simple. Deux cas sont à traiter, comme l'illustre la Table 5.1 : l'inclusion d'une barre unique (expression E_1) et l'inclusion de plusieurs tildes (expression E_2).

$$\begin{aligned} E_1 &= \overbrace{a\overline{(b+\varepsilon)(c+\varepsilon)}d}^{\sim} & L(E_1) &= L(a((b+\varepsilon)(c+\varepsilon) \setminus \varepsilon)d + \varepsilon) \\ &= \overline{\sim}_{(1,4);(2,3)}(a, b+\varepsilon, c+\varepsilon, d) & &= L(a(b+\varepsilon)cd + ab(c+\varepsilon)d + \varepsilon) \end{aligned}$$

$$\begin{aligned} E_2 &= \overline{a\,\overbrace{b}^{\sim}\,\overbrace{c}^{\sim}\,d} & L(E_2) &= L(a((b+\varepsilon)(c+\varepsilon) \setminus \varepsilon)d) \\ &= \overline{\sim}_{(2,2),(3,3);(2,3)}(a, b, c, d) & &= L(a(b+\varepsilon)cd + ab(c+\varepsilon)d) \end{aligned}$$

TABLE 5.1 – *Exemples d'Opérateurs Élémentaires.*

Lemme 5.3 *Soit n un entier supérieur à 0, soit T une liste libre de $\mathcal{S}_n$ et soit (i, f) un couple de $[\![1, n]\!]^2_\leq$ tel que (i, f) ne soit pas dans T ni ne chevauche un élément de T. Soit $E = \overline{\sim}_{T;(i,f)}(E_{1,n})$ une expression ∅-linéaire. On considère les deux cas suivants :*

(1) *S'il existe (i_k, f_k) dans T telle que $i_k \leq i \leq f \leq f_k$, on pose :*

$$\begin{aligned} L' =& L(E_{1\cdots i_1-1}) \cdot L(\widetilde{E_{i_1\cdots f_1}}) \cdot L(E_{f_1+1\cdots i_2-1}) \cdots \\ & L(\widetilde{E_{i_k} \cdots \overline{E_{i\cdots f}} \cdots E_{f_k}}) \cdots \\ & L(E_{f_{\#T-1}+1\cdots i_{\#T}-1}) \cdot L(\widetilde{E_{i_{\#T}\cdots f_{\#T}}}) \cdot L(E_{f_{\#T}+1\cdots n}) \end{aligned}$$

(2) *Si la barre (i, f) inclut m tildes de T avec $m \geq 0$, on pose :*

$$\begin{aligned} L'' =& L(E_{1\cdots i_1-1}) \cdot L(\widetilde{E_{i_1\cdots f_1}}) \cdot L(E_{f_1+1\cdots i_2-1}) \cdots \\ & \overline{L(E_i \cdots \widetilde{E_{i_k\cdots f_k}} \cdots \widetilde{E_{i_{k'}\cdots f_{k'}}} \cdots E_f)} \cdots \\ & L(E_{f_{\#T-1}+1\cdots i_{\#T}-1}) \cdot L(\widetilde{E_{i_{\#T}\cdots f_{\#T}}}) \cdot L(E_{f_{\#T}+1\cdots n}) \end{aligned}$$

*Dans le cas **(1)** (resp. dans le cas **(2)**), on a $L(E) = L'$ (resp. $L(E) = L''$).*

Démonstration. **(I)** Soit w un mot de $L(E)$. Selon la Définition 5.2, il existe une sous-liste libre T' de T telle que T' génère w et telle que w vérifie (i, f) selon T'. Ainsi, selon la Définition 3.3, w admet une décomposition unique $w_1 \cdots w_n$ telle que pour tout k de $[\![1, n]\!]$, $w_k = \varepsilon$ si $k \in \bigcup_{t\in T'} \text{Rangs}(t)$, $w_k \in L(E_k)$ sinon. **(1)** Supposons qu'il existe une tilde $t_k = (i_k, f_k)$ dans T telle que la barre (i, f) soit incluse dans t_k. **(a)** Supposons que t_k est dans T'. Ainsi, $w_{i_k} \cdots w_i \cdots w_f \cdots w_{f_k} = \varepsilon$ et par conséquent w est dans L'. **(b)** Supposons que t_k n'est pas dans T'. Puisque par hypothèse, w vérifie la barre (i, f) selon T', selon la Définition 5.1 et puisqu'aucune tilde t' de T' ne chevauche ou n'inclut la barre b, on a $w_i \cdots w_f \neq \varepsilon$, et par conséquent pour tout k de $[\![i_k, f_k]\!]$, $w_k \in L(E_k)$ et $w_i \cdots w_f \in L(\overline{E_i \cdots E_f})$. Par conséquent, $w \in L'$. **(2)** Supposons que la barre (i, f) inclut l tildes de T avec $l \geq 0$. Puisque par hypothèse, w vérifie la barre (i, f) selon T', selon la Définition 5.1 et puisqu'aucune barre t' de T' ne chevauche ou n'inclut la barre b, on a $w_i \cdots w_f \neq \varepsilon$, et par conséquent pour tout k de $[\![i, f]\!]$, $w_k \in L(E_k)$ et $w_i \cdots w_f \in L(\overline{E_i \cdots E_f})$. Par conséquent, w appartient à L''.

(II) (1) Soit w un mot de L'. Par définition, w de décompose en $w_1 \cdots w_n$ tel que pour tout (i', f') de T, $w_{i'} \cdots w_{f'} \in L(E_{i'\cdots f'}) \cup \{\varepsilon\}$ et tel que pour tout k de $[\![1, n]\!] \setminus \bigcup_{t\in T} \text{Rangs}(t)$, $w_k \in L(E_k)$. Soit T' la liste telle que pour tout (i', f') de $[\![1, n]\!]^2$, (i', f') est dans T' si et seulement si $w_{i'} \cdots w_{f'} = \varepsilon$ et (i', f') est dans T. Par construction, T' est une sous liste de T qui est libre. Par conséquent, la liste T' est libre et selon la Définition 3.3, T' génère w. **(a)** Si (i_k, f_k) est dans T', selon la Définition 5.1, le mot w vérifie (i, f) selon T'. Finalement, selon la Définition 5.2, w est dans $L(E)$. **(b)** Si (i_k, f_k) n'est pas dans T', par définition, $w_{i_k} \cdots w_{f_k} \in L(E_{i_k} \cdots \overline{E_{i\cdots f}} \cdots E_{f_k})$. Par conséquent, $w_{i_k} \cdots w_{f_k} \in L(E_{i_k\cdots f_k})$ et $w_i \cdots w_f \neq \varepsilon$. Selon la Définition 5.1, w vérifie la barre (i, f) selon T'. Finalement, w est dans $L(E)$. **(2)** Soit w un mot de L''. Par définition, w de décompose en $w_1 \cdots w_n$ tel que pour tout (i', f') de T, $w_{i'} \cdots w_{f'} \in L(E_{i'\cdots f'}) \cup \{\varepsilon\}$, tel que pour tout k de $[\![1, n]\!] \setminus \bigcup_{t\in T} \text{Rangs}(t)$, $w_k \in L(E_k)$ et tel que $w_i \cdots w_f \neq \varepsilon$. Soit T' la liste

telle que pour tout (i', f') de $[\![1, n]\!]^2$, (i', f') est dans T' si et seulement si $w_{i'} \cdots w_{f'} = \varepsilon$ et (i', f') est dans T. Par construction, T' est une sous liste de T qui est libre. Par conséquent, la liste T' est libre et selon la Définition 3.3, T' génère w. De plus, puisque $w_i \cdots w_f \neq \varepsilon$, w vérifie (i, f) selon T'. Finalement, w est dans $L(E)$. □

Cas d'une Unique Barre Chevauchant des Tildes d'une Liste Libre :

Comme le montre l'Exemple 5.1, appliquer une tilde sur la liste d'expressions $E_{i,j}$ ajoute le mot vide au facteur $E_{i\cdots j}$ alors qu'appliquer une barre élimine toute occurrence du mot vide apparraissant sur l'intervalle de cette barre. Nous pouvons alors détailler le calcul en nous ramenant à des cas traités dans le Lemme 5.3.

Lemme 5.4 *Soit n un entier supérieur à 0, soit T une liste libre de $\mathcal{S}_n$ et soit (i, f) un couple de $[\![1, n]\!]^2_{\leq}$ tel que (i, f) ne soit pas dans T. Soit $E = \overline{\sim}_{T;(i,f)}(E_{1,n})$ une expression ∅-linéaire. Soit T_+ (resp. T_-) la sous-liste de T des tildes qui chevauchent (resp. qui ne chevauchent pas) la barre (i, f). Considérons les langages L_+ et L_- définis comme suit :*

$$L_+ = \bigcup_{(i_+,f_+)\in T_+} L(\sim_T(E_1, \ldots, E_{i_+-1}, \varepsilon, \ldots, \varepsilon, E_{f_++1}, \ldots, E_n)),$$
$$L_- = L(\overline{\sim}_{T_-;(i,f)}(E_{1,n})).$$

Alors :

$$L(E) = L_- \cup L_+.$$

Démonstration. Soit w un mot de $L(E)$. Selon la Définition 5.2, il existe une sous-liste libre T' de T telle que w est généré par T' et w vérifie (i, f) selon T'. Ainsi, w admet une décomposition $w_1 \cdots w_n$ telle que pour tout (i', f') de T', $w_{i'} \cdots w_{f'} = \varepsilon$ et telle que pour tout k de $[\![1, n]\!] \setminus \bigcup_{t\in T'} \text{Rangs}(t)$, w_k appartient à $L(E_k)$. **(a)** Supposons qu'il existe une tilde $t_+ = (i_+, f_+)$ de T' chevauchant la barre (i, f). Par définition, $w_{i_+} \cdots w_{f_+} = \varepsilon$. De plus, T' est une sous-liste libre de T. Ainsi, w est généré par T' à partir de la liste d'expression $(E_1, \ldots, E_{i_+-1}, \varepsilon, \ldots, \varepsilon, E_{f_++1}, \ldots, E_n)$. Selon la Définition 3.4, w est dans $L(\sim_T(E_1, \ldots, E_{i_+-1}, \varepsilon, \ldots, \varepsilon, E_{f_++1}, \ldots, E_n))$. Finalement, w est dans L_+. **(b)** Supposons qu'il n'existe pas de tildes dans T' chevauchant la barre (i, f). Par définition, on a $T' \subset T_- \subset T$. Ainsi, T' est une sous-liste libre de T_- générant w telle que w vérifie (i, f) selon T'. Selon la Définition 5.2, $w \in L(\overline{\sim}_{T_-;(i,f)}(E_{1,n}))$. Finalement, w est dans L_-.

Soit w un mot de L_+. Alors il existe une tilde (i_+, f_+) de T chevauchant avec (i, f) telle que $w \in L(\sim_T(E_1, \ldots, E_{i_+-1}, \varepsilon, \ldots, \varepsilon, E_{f_++1}, \ldots, E_n))$. Selon la Définition 3.4, il existe une sous-liste libre T' de T telle que T' génère w à partir de la liste $(E_1, \ldots, E_{i_+-1}, \varepsilon, \ldots, \varepsilon, E_{f_++1}, \ldots, E_n)$. Puisque T est une liste libre, on peut supposer sans perte de généralité que la tilde (i_+, f_+) appartient à T' (puisque si T' ne contient pas (i_+, f_+) et génère w, alors la liste libre $T' \cup ((i_+, f_+))$ génère également w). De plus, puisque

$w_{i_+} \cdots w_{f_+} = \varepsilon$, T' génère w à partir de $(E_{1,n})$. Puisque (i_+, f_+) chevauche (i, f), selon la Définition 5.1, w vérifie la barre (i, f) selon T'. Ainsi, T' est une sous-liste libre de T générant w et w vérifie la barre (i, f) selon T'. Selon la Définition 5.2, w est dans $L(E)$.

Soit w un mot de L_-. Selon la Définition 5.2, il existe une sous-liste libre T' de T_- telle que T' génère w et w vérifie (i, f) selon T'. Puisque $T_- \subset T$, T' est une sous-liste libre de T telle que T' génère w et w vérifie (i, f) selon T'. Selon la Définition 5.2, w est dans $L(E)$. □

Cas d'une Liste de Barres Quelconque et d'une Liste de Tildes Libre :

Dans ce cas, tout mot w de $L(\widetilde{\ }_{T;B}(E_{1,n}))$ généré par une sous-liste libre T' de T doit vérifier toute barre de B. Et puisque l'expression est linéaire et qu'ainsi w admet une décomposition unique, nous utilisons une intersection des langages définis dans le Lemme 5.4. Le lemme suivant nous est utile pour démontrer la correction de la formule de ce cas.

Lemme 5.5 *Soit n un entier supérieur à 0, soit $E_{1,n}$ une liste d'expressions sur un alphabet Σ telles que $E_{1\cdots n}$ soit $\emptyset$-linéaire, soit T_1 et T_2 deux listes de $\mathcal{S}_n$ telles que $T_1 \cup T_2$ soit libre, et soit B_1 et B_2 deux listes de $\mathcal{S}_n$ telles que $(B_1 \cup B_2) \cap (T_1 \cup T_2) = \emptyset$. Soit w un mot de Σ^* tel que :*

(1) *T_1 génère w,*

(2) *T_2 génère w,*

(3) *w vérifie toute barre de B_1 selon T_1,*

(4) *w vérifie toute barre de B_2 selon T_2.*

Alors $T_1 \cup T_2$ génère w et w vérifie toute barre de $B_1 \cup B_2$ selon $T_1 \cup T_2$.

Démonstration. **(a)** Montrons que $T_1 \cup T_2$ génère w. Puisque T_1 génère w, selon la Définition 3.3, w admet une décomposition $w_1 \cdots w_n$ telle que pour tout (k, k') de T_1, $w_k \cdots w_{k'} = \varepsilon$ et telle que pour tout entier k de $[\![1, n]\!] \setminus \bigcup_{t \in T_1} \text{Rangs}(t)$, $w_k \in L(E_k)$. Puisque T_2 génère w, selon la Définition 3.3, w admet une décomposition $w_1 \cdots w_n$ telle que pour tout (k, k') de T_2, $w_k \cdots w_{k'} = \varepsilon$ et telle que pour tout k de $[\![1, n]\!] \setminus \bigcup_{t \in T_2} \text{Rangs}(t)$, $w_k \in L(E_k)$. Par conséquent, puisque $T_1 \cup T_2$ est libre, w admet une décomposition $w_1 \cdots w_n$ telle que pour tout (k, k') de $T_1 \cup T_2$, $w_k \cdots w_{k'} = \varepsilon$ et telle que pour tout k de $[\![1, n]\!] \setminus \bigcup_{t \in T_1 \cup T_2} \text{Rangs}(t)$, $w_k \in L(E_k)$. Selon la Définition 3.3, $T_1 \cup T_2$ génère w. **(b)** Montrons que toute barre de $B_1 \cup B_2$ est vérifiée par w selon $T_1 \cup T_2$. Soit (i, f) un couple de $B_1 \cup B_2$. Sans perte de généralité, supposons (i, f) dans B_1. Par hypothèse, w vérifie (i, f) selon T_1. Selon la Définition 5.1, deux cas sont à traiter. Supposons que T_1 contienne une tilde t chevauchant ou incluant (i, f). Alors $T_1 \cup T_2$ contient la tilde t chevauchant ou incluant (i, f). Supposons que $w_i \cdots w_f \neq \varepsilon$. Puisque $E_{1\cdots n}$ est $\emptyset$-linéaire, la décomposition $w_1 \cdots w_n$ est unique. Dans ces deux cas, w vérifie la barre (i, f) selon $T_1 \cup T_2$. □

Lemme 5.6 *Soit n un entier supérieur à 0, soit T et B deux listes disjointes de $\mathcal{S}_n$ telles que T soit libre. Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une expression ∅-linéaire. Le langage dénoté par E se calcule comme suit :*

$$L(\overline{\sim}_{T;B}(E_{1,n})) = \begin{cases} \bigcap\limits_{(i,f)\in B} L(\overline{\sim}_{T;(i,f)}(E_{1,n})) & \text{si } B \neq \emptyset, \\ L(\sim_T(E_{1,n})) & \text{sinon.} \end{cases}$$

Démonstration. Si B est vide, le calcul est correct selon le Lemme 5.1. Considérons alors que B possède au moins élément.

Soit w un mot de $L(E)$. Selon la Définition 5.2, il existe une sous-liste libre T' de T telle que T' génère w et w vérifie toutes les barres de B selon T'. Ainsi, pour toute barre (i,f) de B, T' génère w et w vérifie (i,f) selon T'. Selon la Définition 5.2, w est dans $L(\overline{\sim}_{T;(i,f)}(E_{1,n}))$ pour toute barre (i,f) de B. Par conséquent, w est dans $\bigcap_{(i,f)\in B} L(\overline{\sim}_{T;(i,f)}(E_{1,n}))$.

Soit w un mot de $\bigcap_{(i,f)\in B} L(\overline{\sim}_{T;(i,f)}(E_{1,n}))$. Ainsi, pour toute barre (i,f), selon la Définition 5.2, il existe une sous-liste libre $T'_{(i,f)}$ de T telle que $T'_{(i,f)}$ génère w et w vérifie (i,f) selon $T'_{(i,f)}$. Considérons la liste de tilde $T' = \bigcup_{(i,f)\in B} T'_{(i,f)}$. Par construction de T', puisque chacune des sous-listes $T'_{(i,f)}$ est une sous-liste de T, T' est une sous-liste libre de T. De plus, selon le Lemme 5.5, T' génère w et w vérifie toute barre de B selon T'. Selon la Définition 5.2, w est dans $L(E)$. □

Listes de Tildes et de Barres Quelconques :

Nous donnons finalement l'expression du langage dénoté par une multi-tildes-barres dans le cas général.

Lemme 5.7 *Soit n un entier supérieur à 0, T et B deux listes disjointes de $\mathcal{S}_n$, $\mathcal{T}$ l'ensemble des sous-listes libres de T et $E = \overline{\sim}_{T;(i,f)}(E_{1,n})$ une expression ∅-linéaire. Le langage dénoté par E se calcule comme suit :*

$$L(\overline{\sim}_{T;B}(E_{1,n})) = \bigcup_{T'\in\mathcal{T}} L(\overline{\sim}_{T';B}(E_{1,n})).$$

Démonstration. Soit w un mot de $L(E)$. Selon la Définition 5.2, il existe une sous-liste T' de T telle que T génère w et w vérifie toute barre de B selon T'. Puisque T' est une sous-liste libre de T, T' est dans $\mathcal{T}$. De plus, selon la Définition 5.2, puisque T' est libre et qu'elle est une sous-liste libre de T', $w \in L(\overline{\sim}_{T';B}(E_{1,n}))$. Par conséquent, w est dans $\bigcup\limits_{T'\in\mathcal{T}} L(\overline{\sim}_{T';B}(E_{1,n}))$.

Soit w un mot de $\bigcup\limits_{T'\in\mathcal{T}} L(\overline{\sim}_{T';B}(E_{1,n}))$. Alors il existe une sous-liste libre T' de T telle que w est dans $L(\overline{\sim}_{T';B}(E_{1,n}))$. Selon la Définition 5.2, il existe une sous-liste T'' de T' telle que T'' génère w et w vérifie B selon

T''. Or T'' est une sous-liste libre de T. Ainsi, selon la Définition 5.2, w est dans $L(E)$. □

Corollaire 5.1 *Soit n un entier supérieur à 0, soit T et B deux listes disjointes de $\mathcal{S}_n$. Soit $E = \overline{\sim}_{T;(i,f)}(E_{1,n})$ une expression $\emptyset$-linéaire. Alors :*
le langage dénoté par E est rationnel.

Démonstration. Selon le Lemme 5.7, le Lemme 5.6, le Lemme 5.4 et le Lemme 5.3, le langage d'une expression en multi-tildes-barres s'obtient à partir d'une suite d'unions finies, d'intersections finies, et d'ajout ou d'élimination du mot vide qui sont des opérations rationnelles. Le lien entre ces différents cas est présenté dans la Figure 5.3. □

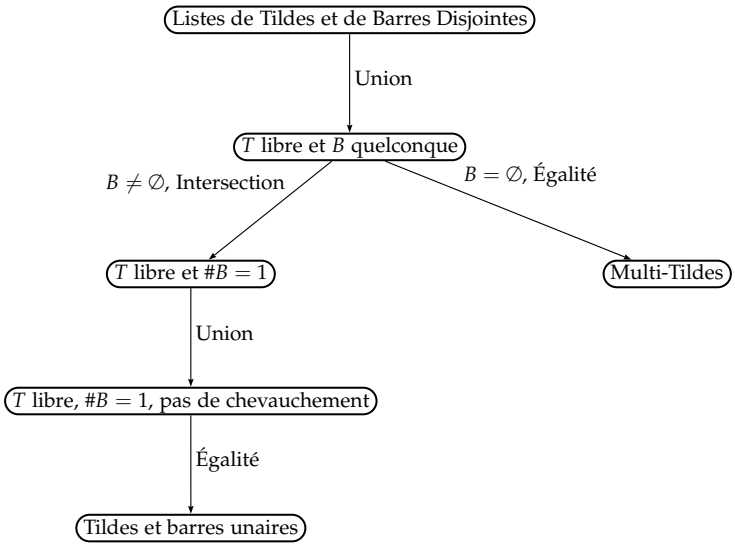


FIGURE 5.3 – *Résumé des Formules du Langage Dénoté par une MTB $\emptyset$-linéaire.*

5.3.3 Une Nouvelle Famille d'Expressions

Selon le Lemme 5.2, les opérateurs de multi-tildes-barres sont compatibles avec la linéarisation. Selon le Corollaire 5.1, le langage dénoté par une multi-tildes-barres $\emptyset$-linéaire est rationnel. Par conséquent, le langage dénoté par une multi-tildes-barres est rationnel. Nous définissons alors une extension des expressions rationnelles.

Définition 5.3 *Une* Expression Rationnelle Augmentée par multi-tildes-barres *(**ERA**) est définie inductivement par :*

$E = \emptyset$	$E = (F + G)$, *avec F et G deux ERA*
$E = \varepsilon$	$E = (F \cdot G)$, *avec F et G deux ERA*
$E = a$, *avec* $a \in \Sigma$,	$E = (F^*)$, *avec F une ERA*

$\overline{\sim}_{T;B}(E_{1,n})$ *, avec T et B deux listes disjointes de $\mathcal{S}_n$ et $E_{1,n}$ une liste d'ERA.*

5.4 La Forme totale

La famille des multi-tildes-barres inclut celle des multi-tildes et celle des multi-barres. Ainsi, deux expressions en multi-tildes-barres basées sur des listes de tildes et/ou de barres différentes appliquées à une même liste d'expressions peuvent dénoter le même langage. Bien que nous puissions étendre la notion d'utilité des éléments pour définir une forme normalisée sur les multi-tildes-barres $\emptyset$-linéaires, nous choisissons d'utiliser ici une forme plus simple à mettre en place, la forme totale. Et nous montrons que deux expressions $\emptyset$-linéaires en multi-tildes-barres en forme totale définies sur des listes de tildes et de barres différentes appliquées à une même liste d'expressions dénotent des langages diffrents.

Définition 5.4 *Une ERA $\overline{\sim}_{T;B}(E_{1,n})$ est* totale *si et seulement si :*

$$T \cup B = [\![1, n]\!]^2_{\leq}.$$

Définition 5.5 *Soit E une ERA. L'ERA E est en* forme totale *(FoTo) si et seulement si toute sous-expression en multi-tildes-barres de E est totale.*

Lemme 5.8 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA $\emptyset$-linéaire totale. Soit $w = w_1 \cdots w_n$ un mot de $(L(E_1) \cup \{\varepsilon\}) \cdots (L(E_n) \cup \{\varepsilon\})$. Les deux conditions suivantes sont équivalentes :*

(1) *w est dans $L(E)$,*

(2) *pour tout facteur ε-maximal $w_i \cdots w_f$ de w, le couple (i, f) est dans T.*

Démonstration. **(1⇒2)** Supposons que $w_i \cdots w_f$ est un facteur ε-maximal de w tel que le couple (i, f) n'est pas dans T. Puisque E est total, le couple (i, f) appartient à B. Montrons que pour toute sous-liste libre T' de T générant w, w ne vérifie pas (i, f). Soit T' une sous-liste libre T' de T générant w. Puisque $w_i \cdots w_f$ est un facteur ε-maximal, T' ne contient pas de tilde chevauchant ou incluant (i, f). Et puisque $w_i \cdots w_f = \varepsilon$, selon la Définition 5.1 w ne vérifie pas (i, f) selon w. Selon la Définition 5.2, $w \notin L(E)$.

(1⇐2) Puisque E est $\emptyset$-linéaire, le mot w admet une unique décomposition $w_1 \cdots w_n$ sur le langage $(L(E_1) \cup \{\varepsilon\}) \cdots (L(E_n) \cup \{\varepsilon\})$ tel que pour tout $k \in [\![1, n]\!]$, $w_k \in L(E_k) \cup \{\varepsilon\})$. Supposons que pour tout facteur ε-maximal $w_i \cdots w_f$ de w, le couple (i, f) est dans T. Considérons la liste T' définie par :

$$T' = ((i, f) \in T \mid w_i \cdots w_f \text{ est un facteur } \varepsilon\text{-maximal dans } w).$$

Par construction, T' génère w. Montrons que w vérifie toute barre de B

selon T'. Soit $b = (i_b, f_b)$ une barre de B. Si pour toute tilde t de T', $\text{Rangs}(t) \cap \text{Rangs}(b) = \emptyset$, par définition de w, $w_{i_b} \cdots w_{f_b} \neq \varepsilon$ et b est vérifiée par w selon T'. Sinon, considérons t une tilde de T' telle que $\text{Rangs}(t) \cap \text{Rangs}(b) \neq \emptyset$. Si t chevauche b ou si t inclut b, b est vérifié par w selon T'. Si b inclut $t = (i_t, f_t)$, puisque $w_{i_t} \cdots w_{f_t}$ est ε-maximal, soit $w_{i_t-1} \neq \varepsilon$, soit $w_{f_t+1} \neq \varepsilon$. Ainsi, $w_{i_b} \cdots w_{f_b} \neq \varepsilon$ et ainsi, w vérifie b selon T'. Par conséquent, T' est une sous-liste de T générant w et telle que w vérifie toute barre de B selon T'. Selon la Définition 5.2, $w \in L(E)$. □

Proposition 5.1 *Soit $E_1 = \overset{\frown}{\overline{\ }}_{T_1;B_1}(F_{1,n})$ et $E_2 = \overset{\frown}{\overline{\ }}_{T_2;B_2}(F_{1,n})$ deux ERA $\emptyset$-linéaires totales. Les deux conditions suivantes sont équivalentes :*

(1) *$B_1 = B_2$ et $T_1 = T_2$,*

(2) *$L(E_1) = L(E_2)$.*

Démonstration. **(1⇒2)** Trivial.
(1⇐2) Puisque les expressions sont en FoTo, $B_1 \neq B_2$ si et seulement si $T_1 \neq T_2$. Soit $t = (i, f)$ une tilde de la différence symétrique de T_1 et T_2. Sans perte de généralité, supposons t dans T_1. Considérons un mot $w = w_1 \cdots w_n$ tel que $w_i \cdots w_f = \varepsilon$ et tel que pour tout k n'appartenant pas à $[\![1, n]\!]$, w_k est dans $L(\overline{F_k})$ (rappelons que $L(\overline{F_k})$ n'est pas vide puisque E_1 et E_2 sont des ERA $\emptyset$-linéaires et que par quotient, les expressions de la liste $F_{1,n}$ sont différentes de ε). Selon le Lemme 5.8, w est dans $L(E_1)$ et w n'est pas dans $L(E_2)$. Par conséquent, $L(E_1) \neq L(E_2)$. □

Nous montrons maintenant que toute ERA admet une ERA en FoTo équivalente. Pour cela, nous étendons la notion de (i, f)-traversabilité aux ERA comme suit :

Définition 5.6 *Soit $E = \overset{\frown}{\overline{\ }}_{T;B}(E_{1,n})$ une ERA $\emptyset$-linéaire, $E'_{1,n}$ une liste d'ERA $\emptyset$-linéaires telle que pour tout k de $[\![1, n]\!]$, $E'_k = \overline{E_k}$ et soit (i, f) un couple de $[\![1, n]\!]^2_{\leq}$. Considérons le langage $L' = L(E'_{1\cdots i-1} \cdot E'_{f+1\cdots n})$.*

L'expression E est (i, f)-traversable si et seulement si $L' \subset L(E)$.

Lemme 5.9 *Soit $E = \overset{\frown}{\overline{\ }}_{T;B}(E_{1,n})$ une ERA $\emptyset$-linéaire, $E'_{1,n}$ une liste d'ERA $\emptyset$-linéaires telle que pour tout k de $[\![1, n]\!]$, $E'_k = \overline{E_k}$ et soit (i, f) un couple de $[\![1, n]\!]^2_{\leq}$. Considérons le langage $L' = L(E'_{1\cdots i-1} \cdot E'_{f+1\cdots n})$. Soit w un mot de L'. Les deux propositions suivantes sont équivalentes :*

(1) *$L' \subset L(E)$,*

(2) *$w \in L(E)$.*

Démonstration. **(1⇒2) :** Trivial.

(1⇐2) : Considérons que le mot w de L' est dans $L(E)$. Montrons que tout mot de L' est dans $L(E)$. Soit w' un mot de L'. Puisque $w \in L(E)$, selon la Définition 5.2, il existe une sous-liste libre T' de T générant w telle

que w vérifie toute barre de B selon T'. Ainsi, w admet une décomposition $w_1 \cdots w_n$ telle que pour tout k de $[\![1, n]\!]$, $w_k = \varepsilon$ s'il existe (i, f) dans T' tel que $k \in [\![i, f]\!]$, $w_k \in L(E_k)$ sinon. Puisque $w_i \cdots w_f$ est le seul facteur ε-maximal de w, que w et w' admettent des décompositions uniques selon L' puisque E est $\emptyset$-linéaire, w' admet une décomposition telle que pour tout k de $[\![1, n]\!]$, $w'_k = \varepsilon$ s'il existe (i, f) dans T' tel que $k \in [\![i, f]\!]$, $w'_k \in L(E_k)$ sinon. Ainsi, w' est généré par T'. Soit b une barre de B. Puisque w vérifie b, soit T' contient une tilde t chevauchant ou incluant b, soit $w_{i_b} \cdots w_{f_b} \neq \varepsilon$. Puisque par construction de w' le seul facteur ε-maximal de w' est $w'_i \cdots w'_f$ et que par définition de w le seul facteur ε-maximal de w est $w_i \cdots w_f$, si $w_{i_b} \cdots w_{f_b} \neq \varepsilon$ alors $w'_{i_b} \cdots w'_{f_b} \neq \varepsilon$. Par conséquent, w' vérifie b selon T'. Ainsi, T' génère w' et w' vérifie b selon T' pour toute barre b de B, et selon la Définition 5.2, $w' \in L(E)$. □

Remarquons que si (i, f) est un couple de T, alors E est (i, f)-traversable. Et si (i, f) est un couple de B alors E n'est pas (i, f)-traversable. La forme totale permet d'établir la réciproque de cette propriété, à savoir que si E est (i, f)-traversable, alors $(i, f) \in T$ et que si E n'est pas (i, f)-traversable, alors $(i, f) \in B$. Nous définissons alors la forme totale E' d'une ERA E et nous montrons que l'expression E' est équivalente à E.

Définition 5.7 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA $\emptyset$-linéaire. Considérons les deux listes T' et B' définies par :*

$$T' = \{(i, f) \in [\![1, n]\!]^2_{\leq} \mid E \text{ est } (i, f)\text{-traversable } \},$$
$$B' = \{(i, f) \in [\![1, n]\!]^2_{\leq} \mid E \text{ n'est pas } (i, f)\text{-traversable } \}.$$

L'expression $E' = \overline{\sim}_{T';B'}(E_{1,n})$ est apellée forme totale de E.

Lemme 5.10 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA $\emptyset$-linéaire et soit E' la forme totale de E. Les deux propositions suivantes sont vérifiées :*

(1) *$T \subset T'$,*

(2) *$B \subset B'$.*

Démonstration. Si (i, f) est un couple de T, alors E est (i, f)-traversable et ainsi selon la Définition 5.7, $(i, f) \in T'$. Si (i, f) est un couple de B, alors E n'est pas (i, f)-traversable et ainsi selon la Définition 5.7, $(i, f) \in B'$. □

Lemme 5.11 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA $\emptyset$-linéaire et soit $w = w_1 \cdots w_n$ un mot de $(L(E_1) \cup \varepsilon) \cdots (L(E_n) \cup \varepsilon)$. Les deux propositions suivantes sont équivalentes :*

(1) *$w \in L(E)$,*

(2) *pour tout $w_i \cdots w_f$ facteur ε-maximal de w, E est (i, f)-traversable.*

Démonstration. **(1⇒2)** Supposons $w \in L(E)$. Alors il existe une sous-liste T' de T telle que T' génère w et w vérifie toute barre de B selon T'. Le mot w est tel que pour tout k de $[\![1, n]\!]$, $w_k = \varepsilon$ s'il existe (i, f) dans T' tel que $k \in [\![i, f]\!]$, $w_k \in L(E_k)$ sinon. Soit (i, f) un couple de $[\![1, n]\!]$ tel que $w_i \cdots w_f$

soit un facteur ε-maximal de w. Soit $w' = w'_1 \cdots w'_n$ un mot tel que pour tout k de $[\![1,n]\!]$, $w'_k = \varepsilon$ si $k \in [\![i,f]\!]$, $w'_k \in L(\overline{E_k})$ sinon. Considérons la liste $U = T' \cap [\![i,f]\!]^2_{\leq}$. Ainsi, le mot w' est tel que pour tout k de $[\![1,n]\!]$, $w'_k = \varepsilon$ s'il existe (i',f') dans U tel que $k \in [\![i',f']\!]$, $w_k \in L(E_k)$ sinon. Par conséquent, U génère w'. Soit b une barre de B. Si b chevauche ou inclus (i,f), ou si $\text{Rangs}(b) \cap [\![i,f]\!] = \emptyset$, par définition de w', $w'_{i_b} \cdots w'_{f_b} \neq \varepsilon$ et w' vérifie b selon U. Supposons b inclus dans (i,f). Puisque w vérifie toute barre de B selon T', w vérifie b selon T'. Par définition, on a :

$$w'_i \cdots w'_{i_b} \cdots w'_{f_b} \cdots w'_f = w_i \cdots w_{i_b} \cdots w_{f_b} \cdots w_f = \varepsilon.$$

Ainsi, il existe une tilde t de T' telle que t chevauche ou inclut b. Par construction de U, $t \in U$. Par conséquent, w' vérifie b selon U. Selon la Définition 5.2, $w' \in L(E)$ et par conséquent selon la Définition 5.6, E est (i,f)-traversable.

(1⇐2) Supposons que pour tout $w_i \cdots w_f$ facteur ε-maximal de w, E est (i,f)-traversable. Soit (i,f) un couple tel que E est (i,f)-traversable. Posons pour tout k de $[\![1,n]\!]$, $E'_k = \overline{E_k}$ et $L' = L(E'_{1\cdots i-1} \cdot E'_{f+1\cdots n})$. Selon la Définition 5.6 et le Lemme 5.9, il existe un mot w' de L' tel que $w' \in L(E)$. Selon la Définition 5.2, il existe une sous-liste libre $T'_{i,f}$ de T telle que $T'_{i,f}$ génère w' et telle que w' vérifie toute barre de B selon $T'_{i,f}$. Considérons la liste T' formée de l'union de ces listes :
$T' = \{(i',f') \in T'_{(i,f)} \mid (i,f) \in [\![1,n]\!]^2 \ \wedge \ w_i \cdots w_f \text{ facteur } \varepsilon\text{-maximal de } w\}$.
Montrons alors que T' génère w et que w vérifie toute barre de B selon T'. Par construction de T', T' génère w, puisque chacun de ses facteurs $w_i \cdots w_f$ ε-maximaux peut être généré par $T'_{(i,f)}$. Soit $b = (i_b, f_b)$ une barre de B. Supposons que w ne vérifie pas b selon T'. Ainsi, T' ne contient pas de tilde chevauchant ou incluant b, et $w_{i_b} \cdots w_{f_b} = \varepsilon$. Soit le facteur $w_{i_b} \cdots w_{f_b}$ est ε-maximal et on pose $(i_b, f_b) = (i,f)$, soit il apparait dans un facteur ε-maximal $w_i \cdots w_{i_b} \cdots w_{f_b} \cdots w_f$. Puisque $w_i \cdots w_f$ est ε-maximal, par hypothèse E est (i,f)-traversable. Ainsi il existe un mot w' de L' tel que $w' \in L(E)$, tel que $T'_{(i,f)}$ génère w' et tel que w' vérifie toute barre de B selon $T'_{(i,f)}$, y compris la barre b. Par conséquent, soit il existe une tilde t dans $T'_{(i,f)}$ chevauchant ou incluant b (contradiction car $t \in T'_{(i,f)} \subset T'$), soit $w_i \cdots w_f \neq \varepsilon$ (contradiction avec $w' \in L'$). Ainsi, w vérifie toute barre de B selon T'. Par conséquent, selon la Définition 5.2, $w \in L(E)$. □

Proposition 5.2 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA $\emptyset$-linéaire et soit E' la forme totale de E. Alors :*

$$L(E) = L(E').$$

Démonstration. Posons $E' = \overline{\sim}_{T';B'}(E_{1,n})$.

Soit w un mot de $L(E)$. Selon la Définition 5.2, il existe une sous-liste libre T'' de T telle que T'' génère w et w vérifie toute barre de B. Le mot w se décompose en $w_1 \cdots w_n$ avec pour tout k de $[\![1,n]\!]$, $w_k = \varepsilon$ s'il existe

(i, f) dans T'' tel que $k \in [\![i, f]\!]$, $w_k \in L(E_k)$ sinon. Selon le Lemme 5.11, pour tout facteur $w_i \cdots w_f$ ε-maximal dans w, E est (i, f)-traversable. Par construction de T', $(i, f) \in T'$. Selon le Lemme 5.8, w est dans $L(E')$.

Soit w un mot de $L(E')$. Selon la Définition 5.2, il existe une sous-liste libre T'' de T' telle que T'' génère w et w vérifie toute barre de B'. Le mot w se décompose en $w_1 \cdots w_n$ avec pour tout k de $[\![1, n]\!]$, $w_k = \varepsilon$ s'il existe (i, f) dans T'' tel que $k \in [\![i, f]\!]$, $w_k \in L(E_k)$ sinon. Selon le Lemme 5.8, pour tout facteur $w_i \cdots w_f$ ε-maximal dans w, $(i, f) \in T'$. Par construction de T', E est (i, f)-traversable. Par conséquent, selon le Lemme 5.11, $w \in L(E)$. □

Corollaire 5.2 *Soit E une ERA. Alors il existe une ERA E' en FoTo telle que $L(E') = L(E)$.*

Démonstration. Selon le Lemme 5.2, l'ERA $\emptyset$-linéarisée $E^\flat$ de E est telle que $\mathrm{h}_E(L(E^\flat) \cap \mathrm{Pos}(E^\flat)^*) = L(E)$. Pour toute sous-expression F de $E^\flat$ en multi-tildes-barres, selon la Proposition 5.2, la forme totale F' de F reconnait le langage $L(F)$. Soit G l'expression obtenue en remplaçant toute sous-expression F de $E^\flat$ en multi-tildes-barres par la forme totale F' de F. Ainsi G est en FoTo et reconnait $L(E^\flat)$. Posons $E' = \mathrm{h}_E(G)$. Selon le Lemme 5.2, E' est équivalente à E et par construction, E' est en FoTo. □

5.5 Conclusion

Les opérateurs de multi-tildes-barres sont des opérateurs rationnels et compatibles avec la linéarisation. La compréhension d'un tel opérateur peut paraître complexe *a priori*. Cependant, pour toute ERA E, l'existence d'une ERA E' équivalente en FoTo permet de décrire l'ensemble des intervalles du point de vue des contraintes induites par la présence des tildes ou des barres, et cela d'une façon unique : puisque tout intervalle $[\![i, f]\!]$ est soit traversable, soit ineffaçable, la forme totale traduit la totalité de ces contraintes appliquées sur un langage. Un opérateur de multi-tildes-barres est ainsi la combinaison des notions de tildes et de barres, introduites dans les chapitres précédents.

Nous montrerons dans le chapitre suivant que la forme totale est également utile pour calculer un automate à $(n + 1)$ états reconnaissant le langage dénoté par une ERA à n lettres.

Chapitre 6

Conversion des ERA en NFA et *vice versa*

Pour apprendre quelque chose aux gens,
il faut mélanger ce qu'ils connaissent avec ce qu'ils ignorent.
Pablo Picasso

Les mathématiques consistent à prouver une chose évidente
par des moyens complexes.
George Pólya

Sommaire

Selon le Théorème de Kleene [30], les expressions rationnelles possèdent le même pouvoir de description que les automates finis. Les algorithmes utilisés pour passer d'un automate A à une expression rationnelle E dénotant $L(A)$ sont de deux types. Les méthodes par système d'équations (Sous-Section 2.2.1), MNY (Sous-Section 2.2.2) ou BMC (Sous-Section 2.2.3) s'appliquent sur un automate A quelconque à n états et produisent une expression rationnelle E dont la taille peut être exponentielle en fonction de n. La méthode CZ (Sous-Section 2.2.4) ne s'applique que sur une partie restreinte de la famille des automates : pour A un automate de Glushkov, l'algorithme produit une expression rationnelle E de largeur $(n-1)$, où n est le nombre d'états de A. C'est cette méthode, calculant

une expression concise, que nous choisissons d'étendre dans ce chapitre. Les méthodes de conversion inverse sont quant à elles définies sur toutes les expressions rationnelles simples. Dans ce chapitre, nous présentons diverses méthodes de conversion entre NFA et ERA. La Section 6.1 introduit les notions d'ERA plates, d'ERA amplifiées et de NFA en ligne. Dans ce chapitre, nous montrons comment calculer un automate A tel que $L(A) = L(E)$ à partir d'une ERA totale E, en utilisant la forme totale définie dans le Chapitre 5. La Section 6.2 présente une construction directe d'un automate A à $(n+1)$ états à partir d'une ERA sans étoile de largeur n. Une extension des fonctions de Glushkov pour les opérateurs de multi-tildes-barres est détaillée dans la Section 6.3, fournissant ainsi une méthode de construction d'un automate A à $(n+1)$ états à partir d'une ERA quelconque de largeur n. Nous donnons un exemple montrant que la méthode de l'automate des follows est applicable à partir de ces fonctions. La Section 6.4 et la Section 6.5 étendent les notions de dérivée, de dérivée partielle et de c-continuation aux opérateurs de multi-tildes-barres, permettant alors de calculer respectivement un DFA, un automate avec au plus $(n+1)$ états ou un automate à $(n+1)$ états reconnaissant le langage dénoté par une ERA de largeur n. La Section 6.6 présente le cas inverse, en étendant l'algorithme CZ (voir Sous-Section 2.2.4 du Chapitre 2), à savoir le calcul à partir d'un automate A de Glushkov étendu possédant $(n+1)$ états d'une ERA E de taille n telle que l'automate des positions de E soit isomorphe à A.

6.1 Quelques Définitions

Dans la suite de cette section, nous utilisons une forme particulière des ERA qui facilite l'étude de la traversabilité des facteurs.

Soit $E = \widetilde{\ }_{T;B}(E_{1,n})$ une ERA sur un alphabet Σ. L'ERA E est *plate* si et seulement si pour tout entier k de l'intervalle $[\![1, n]\!]$, l'ERA E_k est soit un symbole de Σ, soit l'ensemble vide. Une ERA E est en *forme amplifiée* (FoAm) si elle est plate, totale et $\emptyset$-linéaire. Toute ERA $E = \widetilde{\ }_{T;B}(E_{1,n})$ en FoAm sur un alphabet Σ est alors notée $E = \widetilde{\ }_{T;B}(e_1, \ldots, e_n)$ avec pour tout $k \in [\![1, n]\!]$, e_k est un symbole de Σ.

Nous manipulons également des automates acyclique particuliers, les automates en ligne, définis de la façon suivante : un automate acyclique $A = (\Sigma, Q, I, F, \delta)$ est *en ligne* si et seulement pour tout couple d'états (q, q') de Q^2 tel que $q \neq q'$, il existe soit un chemin de q vers q', soit un chemin de q' vers q.

Ainsi, un automate est en ligne si et seulement s'il admet un unique chemin hamiltonien, c'est à dire un chemin passant par tous les états de l'automate une et une seule fois. Les transitions de ce chemin induisent un ordre total sur l'ensemble des états, et en conséquence, tout automate

en ligne admet un unique tri topologique de ses états. Rapelons qu'un tri topologique d'un automate acyclique $A = (\Sigma, Q, I, F, \delta)$ est une bijection τ de Q vers $[\![0, \#Q - 1]\!]$ telle que pour tout couple d'états (q, q') de Q^2, $\tau(q) < \tau(q')$ implique qu'il n'existe pas de chemin de q' vers q.

Un automate standard, homogène et en ligne est appelé *ASHE*. Soit $A = (\Sigma, Q, \{q_0\}, F, \delta)$ un ASHE à $(n+1)$ états et τ son tri topologique. Sans perte de généralité, on peut supposer que A est *trié* selon τ, c'est-à-dire que ses états sont numérotés selon leur position dans le tri topologique τ, la position de l'unique état initial étant 0. Par conséquent, $Q = [\![0, n]\!]$ et $q_0 = 0$. Par exemple, l'automate de la Figure 6.1 est un ASHE trié. Puisque A est standard et homogène, il est possible de définir un étiquetage des transitions selon l'application h_A de Q vers Σ tel que pour tout état q de $Q \setminus \{q_0\}$, $\mathrm{h}_A(q)$ est le symbole de toute transition entrant dans l'état q. Finalement, nous considérons l'*automate jumeau* $A' = (\Sigma', Q, \{0\}, F, \delta')$ *de* A, défini sur l'alphabet $\Sigma' = [\![1, n]\!]$ et tel que δ' est une fonction de $Q \times \Sigma'$ vers 2^Q, avec (p, q, q) dans δ' si et seulement si $(p, \mathrm{h}_A(q), q)$ dans δ. Un automate jumeau est clairement un ASHE trié. Par exemple, l'automate de la Figure 6.2 est l'automate jumeau de l'ASHE trié de la Figure 6.1.

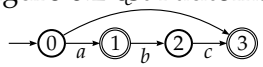


FIGURE 6.1 – *Un ASHE Trié*

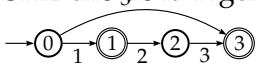


FIGURE 6.2 – *et son Automate Jumeau.*

6.2 Construction Directe pour une ERA sans Étoile

Dans cette section, nous montrons comment calculer un NFA A possédant $(n+1)$ états, à partir d'une ERA E sans étoile de largeur n, tel que $L(A) = L(E)$. Nous montrons tout d'abord que toute ERA E sans étoile peut être transformée en une ERA E' en FoAm reconnaissant le langage $L(E^\flat)$, avec $E^\flat$ l'expression $\emptyset$-linéarisée de E. Puis nous utilisons les propriétés de la FoAm pour calculer un automate reconnaissant $L(E^\flat)$. Enfin, par délinéarisation, nous obtenons alors un NFA reconnaissant $L(E)$.

Tout d'abord, montrons que pour toute ERA E sans étoile, il existe une ERA équivalente n'utilisant que des opérateurs de multi-tildes-barres. Ensuite, nous démontrons l'existence d'une ERA plate équivalente à E.

Lemme 6.1 *Soit E une ERA sans étoile. Il existe une ERA $F = \overline{\sim}_{T;B}(F_{1,n})$ vérifiant les propriétés suivantes :*

(1) *F est de même largeur que E,*

(2) *$L(F) = L(E)$,*

(3) *Tout opérateur utilisé par F est un opérateur de multi-tildes-barres,*

(4) *Aucune occurrence de ε n'apparaît dans F.*

Démonstration. Par induction sur la structure de E. En appliquant les règles de la Table 6.1 récursivement, il est possible de construire une ERA en

multi-tildes-barres F équivalente à E, de même largeur alphabétique, et telle que toute sous-ERA de F est soit un symbole, soit une occurrence de l'ensemble vide, soit une ERA en multi-tildes-barres. □

E'	$\longrightarrow$	F'
$\emptyset$	$\longrightarrow$	$\overline{\sim}_{\emptyset;\emptyset}(\emptyset)$
ε	$\longrightarrow$	$\overline{\sim}_{(1,1);\emptyset}(\emptyset)$
a	$\longrightarrow$	$\overline{\sim}_{\emptyset;\emptyset}(a)$
$E'_1 + E'_2$	$\longrightarrow$	$\overline{\sim}_{(1,2),(2,3);\emptyset}(E'_1, \emptyset, E'_2)$
$E'_1 \cdot E'_2$	$\longrightarrow$	$\overline{\sim}_{\emptyset;\emptyset}(E'_1, E'_2)$

TABLE 6.1 – *Règles de Transformation en MTB.*

Exemple 6.1 *Soit* $E = \overline{\widetilde{(a+b)} \cdot \widetilde{c}}$ *une ERA. Considérons l'expression F suivante :*

$$F = \overline{\sim}_{\emptyset;(1,1)}\big(\overline{\sim}_{\emptyset;\emptyset}\big(\overline{\sim}_{(1,1);\emptyset}\big(\overline{\sim}_{(1,2),(2,3);\emptyset}(a, \emptyset, b)\big),\ \overline{\sim}_{(1,1);\emptyset}(c)\big)\big).$$

L'expression F est obtenue par les transformations de la Table 6.1 appliquées sur l'expression E. Les arbres syntaxiques des expressions sont représentés Figure 6.3.

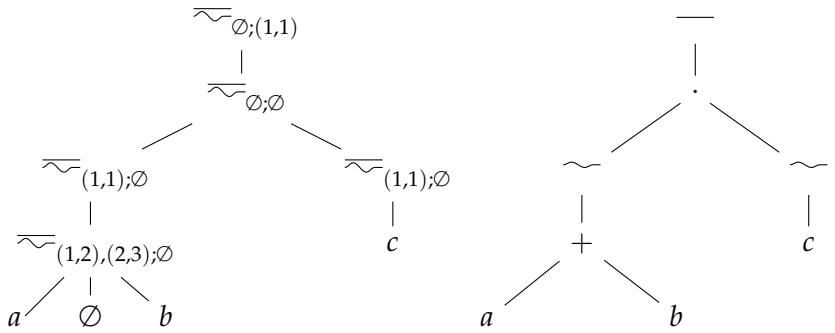


FIGURE 6.3 – *Les Arbres Syntaxiques de F et de E.*

Lemme 6.2 *Soit* $E = \overline{\sim}_{T;B}(E_{1,n})$ *une ERA telle que tout opérateur de E soit un opérateur de multi-tildes-barres et telle qu'aucune occurrence du mot vide n'apparaisse dans E. Alors il existe une ERA F plate, de même largeur que E et telle que* $L(F) = L(E)$.

Démonstration. Supposons qu'il existe une sous-ERA $E' = \overline{\sim}_{T';B'}(E'_{1,n'})$ de E telle qu'il existe un entier k dans $[\![1, n']\!]$ vérifiant $E'_k = \overline{\sim}_{T'';B''}(E''_{1,m})$. Considérons les listes suivantes :

$$T_1 = \big((i,f) \in T' \mid f < k\big),$$

$$T_2 = \begin{cases} \big((i+k-1, f+k-1) \mid (i,f) \in T''\big) & \text{si } (k,k) \notin T' \cup B', \\ \big((i+k-1, f+k-1) \mid (i,f) \in T''\big) \setminus (k, k+n'-1) & \text{sinon,} \end{cases}$$

$$T_3 = \begin{cases} \big((i, f+m-1) \mid (i,f) \in T' \ \wedge\ i \le k \le f\big) \\ \cup\ \big((i+m-1, f+m-1) \mid (i,f) \in T' \ \wedge\ k < i\big), \end{cases}$$

$B_1 = \big((i,f) \in B' \mid f < k\big)$,
$B_2 = \begin{cases} \big((i+k-1, f+k-1) \mid (i,f) \in B''\big) & \text{si } (k,k) \notin T' \cup B', \\ \big((i+k-1, f+k-1) \mid (i,f) \in B''\big) \setminus (k, k+n'-1) & \text{sinon,} \end{cases}$
$B_3 = \begin{cases} \big((i, f+m-1) \mid (i,f) \in B' \ \wedge\ i < k \leq f\big) \\ \cup \big((i+m-1, f+n'-1) \mid (i,f) \in B \ \wedge\ k < i\big). \end{cases}$

Pour tout l de $[\![1, n'+m-1]\!]$, posons $F'_l = \begin{cases} E'_l & \text{si } l < k, \\ E''_{l-k+1} & \text{si } k \leq l \leq k+m-1, \\ E'_{l-m+1} & \text{sinon.} \end{cases}$

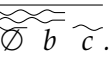
Considérons alors l'expression $F' = \overline{\sim}_{T_1 \cup T_2 \cup T_3; B_1 \cup B_2 \cup B_3}\big(F'_{1,m+n'-1}\big)$. On peut vérifier que E' et F' sont équivalentes, et ainsi on peut remplacer E' par F' dans E.

En appliquant récursivement cette transformation à toute sous-ERA en multi-tildes-barres de E, on obtient une expression F plate, équivalente à E et de même largeur. □

Exemple 6.2 *Soit F l'expression de l'Exemple 6.1 et G l'expression suivante :*

$$G = \overline{\sim}_{(1,2),(1,3),(2,3),(4,4);(1,4)}\big(a, \emptyset, b, c\big).$$

L'ERA plate G est équivalente à F. Les arbres syntaxiques des expressions G et F sont représentés Figure 6.4. Remarquons que les représentations graphiques de G et de F sont identiques, à savoir :



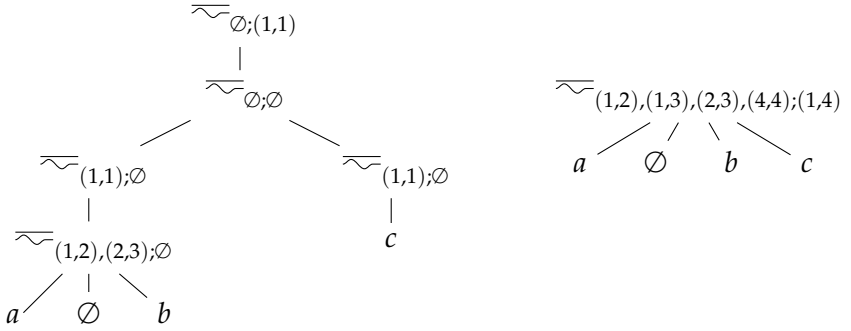


FIGURE 6.4 – *Les Arbres Syntaxiques de F et de G.*

Le Corollaire 5.2 permet de transformer toute expression plate E' en une expression plate et totale E'' équivalente. L'expression $\emptyset$-linéarisée de E'', notée $E''^\flat$, est en FoAm, et les propriétés vérifiées par cette forme nous permettent de construire un automate reconnaissant $L(E''^\flat)$ très facilement.

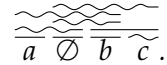
Proposition 6.1 *Soit E une ERA sans étoile. Soit $E^\flat$ l'expression $\emptyset$-linéarisée de E. Alors il existe une expression E' en FoAm équivalente à $E^\flat$. L'ERA E' est apellée* FoAm *de E.*

Démonstration. Selon le Lemme 6.1 et le Lemme 6.2, il existe une ERA F

plate et équivalente à E. Soit F' l'expression $\emptyset$-linéarisée de F. Puisque les opérateurs de multi-tildes-barres sont compatibles avec la linéarisation, les langages $L(F)$ et $\mathrm{h}(L(F') \cap \mathrm{Pos}(E))$ sont égaux. Soit E' la FoTo de F'. Selon le Corollaire 5.2, les expressions F' et E' sont équivalentes. Ainsi, E' est en FoAm et reconnaît le langage $L(E^\flat)$. □

Exemple 6.3 *Considérons les ERA E, F et G de l'Exemple 6.1 et de l'Exemple 6.2. Considérons l'expression H suivante et sa représentation graphique :*

$$H = \overline{\sim}_{(1,2),(1,3),(2,3),(2,4),(4,4);(1,1),(1,4),(2,2),(2,4),(3,3)}(a, \emptyset, b, c),$$



Les expressions E, F, G et H dénotent le même langage. L'expression linéarisée de H, définie par

$$H' = \overline{\sim}_{(1,2),(1,3),(2,3),(2,4),(4,4);(1,1),(1,4),(2,2),(2,4),(3,3)}(1, I, 2, 3)$$

est la FoAm de E.

Définition 6.1 *Soit E une ERA sans étoile et $E' = \overline{\sim}_{T';B'}(e_1, \ldots, e_n)$ la FoAm de E. Considérons le NFA $A' = (\Sigma', Q', I', F', \delta')$ défini par :*

(-) $\Sigma' = \{e_1, \ldots, e_n\}$,
(-) $Q' = \Sigma' \cup \{e_0\}$,
(-) $I' = \{e_0\}$,
(-) $F' = \{e_n\} \cup \{e_k \mid (k+1, n) \in T'\}$,
(-) $\delta' = \{(e_k, e_{k'}, e_{k'}) \in Q' \times \Sigma' \times Q' \mid k' = k+1 \ \vee \ (k+1, k'-1) \in T'\}$.

Le NFA A' est appelé automate caractéristique de *E.*

Lemme 6.3 *Soit E une ERA sans étoile et soit A' l'automate caractéristique de E. L'automate A' est homogène, standard et en ligne.*

Démonstration. Par construction de l'automate caractéristique selon la Définition 6.1. □

Proposition 6.2 *Soit E une ERA sans étoile, $E' = \overline{\sim}_{T';B'}(e_1, \ldots, e_n)$ la FoAm de E et A' l'automate caractéristique de E. Soit $w = w_1 \cdots w_n$ un mot tel que pour tout entier k de $[\![1, n]\!]$, w_k est dans $\{\varepsilon, e_k\}$. Les deux propositions suivantes sont équivalentes :*

(1) *w est dans $L(A')$,*
(2) *w est dans $L(E')$.*

Démonstration. Posons $A' = (\Sigma', Q', I', F', \delta')$.

(1)⇐(2) Supposons que w est dans $L(E')$. Par construction de A', pour tout entier k de $[\![0, n-1]\!]$, e_{k+1} est dans $\delta'(e_k, e_{k+1})$. Puisque E' est une expression totale, selon le Lemme 5.8, pour tout facteur ε-maximal $w_i \cdots w_f$, le couple (i, f) est dans T'. Par construction de A', cela implique que e_{f+1} est dans $\delta'(e_{i-1}, e_{f+1})$ si $f \neq n$ et que e_{i-1} est dans F' sinon. Ainsi w est l'étiquette d'un chemin réussi de A'.

(1)⇒(2) Soit w l'étiquette d'un chemin réussi de A'. Pour tout facteur ε-maximal $w_i \cdots w_f$, soit le chemin contient une transition $(e_{i-1}, e_{f+1}, e_{f+1})$ si $f \neq n$, soit l'état e_{i-1} est final, et par construction de A', (i, f) est dans T'. Selon le Lemme 5.8, w est dans $L(E')$. □

Exemple 6.4 *Considérons l'ERA E et sa FoAm H′ de l'Exemple 6.3. L'automate caractéristique de E, calculé à partir de H′ est l'automate représenté Figure 6.5.*

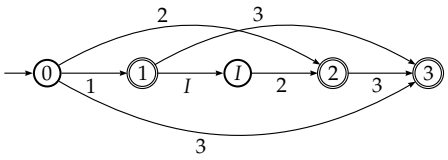


FIGURE 6.5 – *L'Automate Caractéristique de E.*

Finalement, pour obtenir un automate reconnaissant le langage de l'expression de départ, nous délinéarisons l'automate caractéristique A en le projetant de $\mathrm{Pos}(E)$ vers Σ et en éliminant tout élément de $\mathrm{Pos}_\emptyset(E)$.

Définition 6.2 *Soit E une ERA sans étoile sur un alphabet Σ. Soit $A' = (\Sigma', Q', I', F', \delta')$ l'automate caractéristique de E. Soit le NFA $A = (\Sigma, Q, I, F, \delta)$ défini par :*

(-) $Q = \mathrm{Pos}(E) \cup \{0\}$,
(-) $I = \{0\}$,
(-) $F = F' \cap \mathrm{Pos}(E)$,
(-) $\delta = \{(q, \mathrm{h}_E(e_k), q') \in Q \times \Sigma \times Q \mid (q, e_k, q') \in \delta'\}$.

Le NFA A est appelé automate projeté de *E.*

Proposition 6.3 *Soit E une ERA sans étoile de largeur n et soit A l'automate projeté de E. Le NFA A possède $(n+1)$ états et reconnait le langage dénoté par E.*

Démonstration. Soit E' la FoAm de E et $A' = (\Sigma', Q', I', F', \delta')$ l'automate caractéristique de E. Posons $A = (\Sigma, Q, I, F, \delta)$.

Par construction, on a $\#Q = \#\mathrm{Pos}(E) + 1 = n + 1$.

Soit w un mot de $L(E)$. Par construction de E', il existe w' de $L(E')$ tel que $\mathrm{h}_E(w') = w$. Trivialement, w' est dans $\mathrm{Pos}(E)^*$. Selon la Proposition 6.2, w' est dans $L(A')$. Ainsi, il existe un chemin réussi étiqueté par w' dans A' ne passant que par des états e_k de Q' tels que $\mathrm{h}_E(e_k) \neq \emptyset$. Par conséquent, par construction de A, il existe un chemin réussi étiqueté par $\mathrm{h}_E(w')$ dans A. Puisque $\mathrm{h}_E(w') = w$, w est dans $L(A)$.

Soit w un mot de $L(A)$. Par construction de A, il existe un chemin réussi étiqueté par w' dans A' tel que $\mathrm{h}_E(w') = w$ et tel que w' est dans $\mathrm{Pos}(E)^*$. Selon la Proposition 6.2, w' est dans $L(E')$. Selon la Définition 4.2, les langages $L(E)$ et $\mathrm{h}_E(L(E') \cap \mathrm{Pos}(E)^*)$ sont égaux. Ainsi, $\mathrm{h}_E(w') = w$ est dans $L(E)$. □

Exemple 6.5 *Considérons l'ERA E et son automate caractéristique A′ de l'Exemple 6.4. L'automate projeté de E, calculé à partir de A′ est l'automate représenté Figure 6.6.*

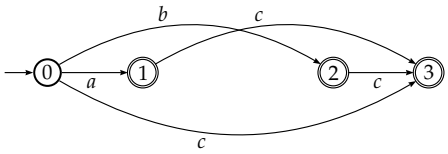


FIGURE 6.6 – *L'Automate Projeté de E.*

6.3 Extensions des Fonctions De Glushkov

Dans cette section, nous étendons le calcul des fonctions de Glushkov aux multi-tildes-barres en FoTo.

Définition 6.3 *Soit* $E = \overline{\sim}_{T;B}(E_{1,n})$ *une ERA ∅-linéaire en FoTo, l un entier de* $[\![1,n]\!]$ *et x un élément de* $\mathrm{Pos}(E_l)$. *Les fonctions de Glushkov de l'ERA E sont :*

$$\textbf{(6.3.1)}\ \mathrm{Pos}(E)=\bigcup_{k=1}^{n}\mathrm{Pos}(E_k),$$

$$\textbf{(6.3.2)}\ \mathrm{Null}(E)=\begin{cases}\varepsilon & si\ (1,n)\in T,\\ \emptyset & sinon,\end{cases}$$

$$\textbf{(6.3.3)}\ \mathrm{First}(E)=\begin{cases}\mathrm{First}(E_1)\cup\\ \{\mathrm{First}(E_k)\mid k\in[\![2,n]\!]\ \wedge\ (1,k-1)\in T\},\end{cases}$$

$$\textbf{(6.3.4)}\ \mathrm{Last}(E)=\begin{cases}\mathrm{Last}(E_n)\cup\\ \{\mathrm{Last}(E_k)\mid k\in[\![1,n-1]\!]\ \wedge\ (k+1,n)\in T\},\end{cases}$$

$$\textbf{(6.3.5)}\ \mathrm{Follow}(x,E)=\begin{cases}\mathrm{Follow}(x,E_l) & si\ l=n\\ & \vee\ x\notin\mathrm{Last}(E_l),\\ \mathrm{Follow}(x,E_l)\cup\mathrm{First}(x,E_{l+1})\cup & \\ \{\mathrm{First}(E_{l'})\mid l'\in[\![l+2,n]\!]\wedge & \\ (l+1,l'-1)\in T\} & sinon.\end{cases}$$

Proposition 6.4 *Soit* $E = \overline{\sim}_{T;B}(E_{1,n})$ *une ERA ∅-linéaire en FoTo obtenue par ∅-linéarisation d'une ERA E′, et soit* G_E *son automate de Glushkov. Alors :*

$$L(E) = L(G_E).$$

Démonstration. La définition des fonctions de Glushkov sur les opérateurs $+$, $\cdot$ et * induisent :
(P1) $x\in\Sigma_E \Leftrightarrow x\in\mathrm{Pos}(E)$,
(P2) $\mathrm{Null}(E)=\begin{cases}\{\varepsilon\} & \text{si } \varepsilon\in L(E),\\ \emptyset & \text{sinon,}\end{cases}$
(P3) $\exists w = x\cdot w'\in L(E)\Leftrightarrow x\in\mathrm{First}(E)$,
(P4) $\exists w = w'\cdot x\in L(E)\Leftrightarrow x\in\mathrm{Last}(E)$,
(P5) $\exists w = w'\cdots x\cdot x'\cdots w''\in L(E)\Leftrightarrow x'\in\mathrm{Follow}(x,E)$.

Montrons que les propositions **(P1)** à **(P5)** sont toujours vraies lors de l'extension aux opérateurs de multi-tildes-barres.
(P1) Pos **:** $x\in\Sigma_E \Leftrightarrow \exists k\in[\![1,n]\!]\mid x\in\Sigma_{E_k}\ \Leftrightarrow x\in\mathrm{Pos}(E)$.

(P2) Null **:** $\varepsilon \in L(E) \Leftrightarrow E$ est $(1,n)$-traversable $\Leftrightarrow (i,f) \in T$.

(P3) First **:** ($\Rightarrow$) Soit $w = x \cdot w'$ un mot de $L(E)$. Selon la Définition 5.2, w admet une décomposition $w_1 \cdots w_n$ telle que pour tout entier k de $[\![1,n]\!]$, w_k est dans $L(E_k) \cup \{\varepsilon\}$. Puisque w est linéaire, il existe un unique l de $[\![1,n]\!]$ tel que $w_l = x \cdot w'_l \in L(E_l)$. Par définition, $x \in \text{First}(E_l)$. Si $l = 1$, selon la Définition 6.3.3, $\text{First}(E_1) \subset \text{First}(E)$. Si $l \neq 1$, $w_1 \cdots w_{l-1}$ est un facteur ε-maximal. Selon le Lemme 5.8, $(1, l-1)$ est dans T. Selon la Définition 6.3.3, $\text{First}(E_l) \subset \text{First}(E)$. Par conséquent, x est dans $\text{First}(E)$.

($\Leftarrow$) Soit x dans $\text{First}(E)$. Alors il existe un entier l de $[\![1,n]\!]$ tel que x est dans $\text{First}(E_l)$ et soit $l = 1$, soit $l \neq 1$ et $(1, l-1)$ est dans T. Par définition, il existe un mot $w_l = x \cdot w'_l$ dans $L(E_l)$. Considérons le mot $w = w_l \cdot w_{l+1} \cdots w_n$ tel que pour tout k de $[\![l,n]\!]$, $w_k \in L(\overline{E_k})$. Si $l = 1$, w n'a aucun facteur ε-maximal par construction, et ainsi selon le Lemme 5.8, w est dans $L(E)$. Si $l \neq 1$, le facteur $w_1 \cdots w_{l-1}$ est le seul facteur ε-maximal de w. Puisque $(1, l-1)$ est dans T, selon le Lemme 5.8, w est dans $L(E)$.

(P4) Last **:** La preuve est similaire à (P3) en raisonnant sur le dernier symbole des mots.

(P5) Follow **:** Soit l l'entier de $[\![1,n]\!]$ tel que $x \in \text{Pos}(E_l)$.

($\Rightarrow$) Soit $w = w' \cdot x \cdot x' \cdot w''$ un mot de $L(E)$. Selon la Définition 5.2, w admet une décomposition $w_1 \cdots w_n$ telle que pour tout entier k de $[\![1,n]\!]$, w_k est dans $L(E_k) \cup \{\varepsilon\}$. Soit l' l'entier de $[\![1,n]\!]$ tel que $x' \in \text{Pos}(E_{l'})$. Trois cas sont à traiter. **(a)** Si $l = l'$, alors $w_l = w'_l \cdot x \cdot x' \cdot w''_l$. Puisque w_l est dans $L(E_l)$, par définition, x' est dans $\text{Follow}(x, E_l)$. Selon la Définition 6.3.5, x' est dans $\text{Follow}(x, E)$. **(b)** Si $l' = l+1$, $w_l = w'_l \cdot x$ et $w_{l'} = x' \cdot w''_{l'}$. Par définition, x est dans $\text{Last}(E_l)$ et x' est dans $\text{First}(E_{l'})$. Selon la Définition 6.3.5, x' est dans $\text{Follow}(x, E)$. **(c)** Supposons que $l \neq l'$ et $l' \geq l+2$. Alors $w_l = w'_l \cdot x$, $w_{l'} = x' \cdot w''_{l'}$ et $w_{l+1} \cdots w_{l'-1}$ est un facteur ε-maximal dans w. Selon le Lemme 5.8, $(l+1, l'-1)$ est dans T. Selon la Définition 6.3.5, x' est dans $\text{Follow}(x, E)$.

($\Leftarrow$) Soit x' un élément de $\text{Follow}(x, E)$. Selon la Définition 6.3.5, trois cas sont à traiter. **(a)** Supposons x' est dans $\text{Follow}(x, E_l)$. Par hypothèse, il existe un mot $w_l = w'_l \cdot x \cdot x' \cdot w''_l$ dans $L(E_l)$. Soit $w = w_1 \cdots w_l \cdots w_n$ un mot tel que pour tout k de $[\![1,n]\!]$, $w \in L(\overline{E_k})$. Le mot w ne possède aucun facteur ε-maximal et selon le Lemme 5.8, w est dans $L(E)$. **(b)** Supposons x est dans $\text{Last}(E_l)$ et x' est dans $\text{Follow}(x, E_{l+1})$. Par hypothèse, il existe $w_l = w'_l \cdot x$ dans $L(E_l)$ et $w_{l+1} = x' \cdot w'_{l+1}$ dans $L(E_{l+1})$. Considérons le mot $w = w_1 \cdots w_l \cdot w_{l+1} \cdots w_n$ tel que pour tout k de $[\![1,n]\!]$, $w \in L(\overline{E_k})$. Le mot w ne possède aucun facteur ε-maximal et selon le Lemme 5.8, w est dans $L(E)$. **(c)** Supposons que x est dans $\text{Last}(E_l)$ et qu'il existe l' dans $[\![1,n]\!]$ tel que x' est dans $\text{First}(E_{l'})$ et $(l+1, l'-1)$ dans T. Par hypothèse, il existe $w_l = w'_l \cdot x$ dans $L(E_l)$ et $w_{l+1} = x' \cdot w'_{l+1}$ dans $L(E_{l+1})$. Considérons un mot $w = w_1 \cdots w_l \cdot w_{l+1} \cdots w_n$ tel que pour tout k de $[\![1,n]\!]$, $w \in L(\overline{E_k})$. Le

facteur $w_{l+1} \cdots w_{l'-1}$ est le seul facteur ε-maximal de w, et par hypothèse, $(l+1, l'-1)$ est dans T. Selon le Lemme 5.8, w est dans $L(E)$. □

Exemple 6.6 *Soit* $E = \overbrace{(a+b)}^{\sim} \cdot a^*$ *une ERA totale et* $E' = \overbrace{(1+2)}^{\sim} \cdot 3^*$ *l'expression ∅-linéarisée de E. Les fonctions de Glushkov de l'expression E ainsi que celles étendues à la position 0 sont données Table 6.2. L'automate des positions de E et l'automate des follows de E sont représentés Figure 6.7 et Figure 6.8.*

$\mathrm{Pos}(E) = \{1,2,3\}$	$\mathrm{Pos}_0(E) = \{0,1,2,3\}$
$\mathrm{Null}(E) = \emptyset$	
$\mathrm{First}(E) = \{1,2,3\}$	$\mathrm{Follow}_0(0,E) = \{1,2,3\}$
$\mathrm{Last}(E) = \{1,2,3\}$	$\mathrm{Last}_0(E) = \{1,2,3\}$
$\mathrm{Follow}(1,E) = \{3\}$	$\mathrm{Follow}_0(1,E) = \{3\}$
$\mathrm{Follow}(2,E) = \{3\}$	$\mathrm{Follow}_0(2,E) = \{3\}$
$\mathrm{Follow}(3,E) = \{3\}$	$\mathrm{Follow}_0(3,E) = \{3\}$

TABLE 6.2 – *Les Fonctions de Glushkov de l'Expression E.*

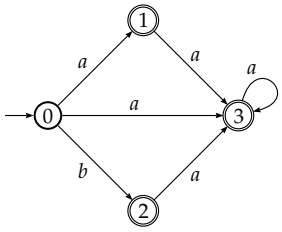


FIGURE 6.7 – *L'Automate des Positions de E.*

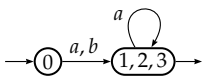


FIGURE 6.8 – *L'Automate des Follows de E.*

6.4 Dérivées et Dérivées Partielles d'une ERA

Dans cette sous-section, nous étendons les formules de dérivation et de dérivation partielle aux ERA en multi-tildes-barres, et nous montrons que les propriétés nécessaires à la construction d'un automate fini sont conservées, à savoir le lien entre le quotient d'un langage et la dérivée d'une expression le dénotant, ainsi que le caractère fini du cardinal de l'ensemble des dérivées et de celui des dérivées partielles.

Pour cela, nous allons découper les listes sur lesquelles porte l'opérateur de multi-tildes-barres. Soit n un entier positif et soit S une liste de $\mathcal{S}_n$. Soit k un entier de $[\![1, n]\!]$. On note la liste $S_{\leq k}$ (resp. $S_{\geq k}$) la liste $S_{\leq k} = ((i,f) \in S \mid f \leq k)$ (resp. $S_{\geq k} = ((i-k+1, f-k+1) \in S \mid i \geq k)$). Remarquons qu'une renumérotation est réalisée lors du calcul de $S_{\geq k}$, correspondant à une translation.

Exemple 6.7 *Considérons la liste* $S = ((1,1),(1,2),(1,3),(1,4),(2,3),(3,4),(4,4))$ *de* $\mathcal{S}_4$ *représentée graphiquement dans la Figure 6.9. Ainsi, les listes* $S_{\leq 2}$ *et* $S_{\geq 3}$ *sont définies par*

$$S_{\leq 2} = ((1,1),(1,2)),$$
$$S_{\geq 3} = ((3-3+1,4-3+1),(4-3+1,4-3+1)) = ((1,2),(2,2)).$$

Ces deux listes sont représentées dans la Figure 6.10.

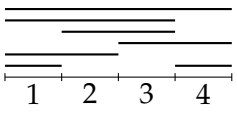


FIGURE 6.9 – *La Suite S.*

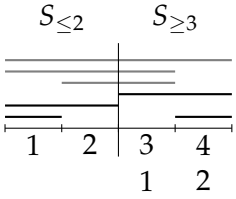


FIGURE 6.10 – *Les Suites* $S_{\leq 2}$ *et* $S_{\geq 3}$.

Nous montrons alors comment calculer le quotient du langage dénoté par une multi-tildes-barres totale, puis nous étendons les formules de dérivation (Définition 2.5) et de dérivation partielle (Définition 2.7) aux ERA.

Proposition 6.5 *Soit* $E = \overline{\sim}_{T;B}(E_{1,n})$ *une ERA totale sur un alphabet* Σ *et a une lettre de* Σ. *Posons :*

$$L' = \begin{cases} a^{-1}(L(E_1)) \cdot L(\overline{\sim}_{T_{\geq 2};B_{\geq 2}}(E_{2,n})) \cup \\ \bigcup_{k \in [\![2,n]\!] \wedge (1,k-1) \in T} a^{-1}(L(E_k)) \cdot L(\overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n})). \end{cases}$$

On a : $L' = a^{-1}(L(E))$.

Démonstration. Par induction sur la structure des ERA, sur le nombre d'opérateurs et sur le nombre de couples des opérateurs à contraintes. Selon le Lemme 1.4, pour toute expression rationnelle simple, $a^{-1}(F)$ est calculable par induction. Nous étendons ces formules sur les ERA.

Soit w un mot de L'. Montrons que $a \cdot w$ est dans $L(E)$. Par hypothèse, il existe un entier k dans $[\![1,n]\!]$ tel que w soit dans $a^{-1}(L(E_k)) \cdot L(\overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}))$ et tel que si $k \neq 1$, alors le couple $(1,k-1)$ est dans T. Ainsi, w se décompose en $w_k \cdot w'$ avec w_k dans $a^{-1}(L(E_k))$ et w' dans $L(\overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}))$. Par définition, $w_k \in a^{-1}(L(E_k))$ implique $a \cdot w_k \in L(E_k)$. Selon la Définition 5.2, w' se décompose en $w'_{k+1} \cdots w'_n$ et il existe une sous liste libre U de $T_{\geq k+1}$ générant w' telle que w' vérifie toute barre de $B_{\geq k+1}$ selon U. Considérons alors le mot $v = a \cdot w_k \cdot w' = a \cdot w$. Par construction, v se décompose en $v_k \cdots v_n$ avec $v_k = a \cdot w_k$ et $v_{k'} = w'_{k'}$ pour tout k' supérieur à k. **(I)** Montrons qu'il existe une sous-liste T' de T générant v. **(a)** Si $k = 1$, considérons la liste $T' = ((i+k, f+k) \mid (i,f) \in U)$. Puisque $v_1 = a \cdot w_1 \neq \varepsilon$ et $v_1 \in L(E_1)$, T' est une sous-liste libre de T générant v. **(b)** Si $k \neq 1$, considérons la liste T' définie par

$$T' = ((i+k, f+k) \mid (i,f) \in U) \cup (1,k-1).$$

Par hypothèse, $(1,k-1)$ est dans T. Puisque $v_k = a \cdot w_k \neq \varepsilon$ et $v_k \in L(E_k)$,

T' est une sous-liste libre de T générant v. **(II)** Montrons que v vérifie toute barre de B selon T'. Par hypothèse, w' vérifie toute barre de $B_{\geq k+1}$ selon U. Par définition de $B_{\geq k+1}$, la barre $(i_b + k, f_b + k)$ est dans B si et seulement si (i_b, f_b) est dans $B_{\geq k+1}$. Montrons alors que si w' vérifie la barre (i_b, f_b) de $B_{\geq k+1}$ selon U, v vérifie la barre $(i_b + k, f_b + k)$ de B selon T'. Selon la Définition 5.1, soit U contient une tilde (i_t, f_t) chevauchant ou incluant (i_b, f_b) et par construction T' contient une tilde $(i_t + k, f_t + k)$ qui inclut ou chevauche $(i_b + k, f_b + k)$, soit $w'_{i_b+k} \cdots w'_{f_b+k} \neq \varepsilon$ et ainsi $v'_{i_b+k} \cdots w'_{v_b+k} \neq \varepsilon$. De plus, toute barre commençant en un rang inférieur à k est vérifiée soit en étant chevauchée ou soit en étant incluse dans le couple $(1, k-1)$ de T', et toute barre commençant en k est vérifiée puisque $v_k \neq \varepsilon$. Par conséquent, v vérifie toute barre de B selon T'. Finalement, selon la Définition 5.2, $v = a \cdot w$ est dans $L(E)$.

Soit w un mot de $a^{-1}L(E)$. Montrons que $w \in L'$. Par définition, le mot $v = a \cdot w$ est dans $L(E)$. Ainsi, selon la Définition 5.2, il existe une sous-liste libre T' de T telle que $v = v_1 \cdots v_n$, avec pour tout k de $[\![1, n]\!]$, $v_k = \varepsilon$ s'il existe (i, f) dans T' tel que k est dans $[\![i, f]\!]$, $v_k \in L(E_k)$ sinon. Ainsi il existe k dans $[\![1, n]\!]$ tel que $v_k = a \cdot w'_k \in L(E_k)$. Si $k \neq 1$, puisque $v_1 \cdots v_{k-1}$ est un facteur ε-maximal et que T' génère v, $(1, k-1)$ est dans T'. Par hypothèse d'induction, $w'_k = a^{-1}(v_k)$ est dans $a^{-1}(L(E_k))$. Considérons le mot $w' = v_{k+1} \cdots v_n$ et montrons qu'il appartient à $L(\overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}))$. Considérons la liste U définie par :

$$U = ((i-k, f-k) \mid (i, f) \in T' \setminus \{(1, k-1)\}.$$

Par construction, U est une sous-liste libre de $T_{\geq k+1}$, et puisque T' génère v, alors U génère w'. Montrons que w' vérifie toute barre de $B_{\geq k+1}$ selon U. Par hypothèse, v vérifie toute barre de B selon T'. Par définition de $B_{\geq k+1}$, la barre $(i_b + k, f_b + k)$ est dans B si et seulement si (i_b, f_b) est dans $B_{\geq k+1}$. Montrons alors que si v vérifie la barre $(i_b + k, f_b + k)$ de B selon T', alors w' vérifie la barre (i_b, f_b) de $B_{\geq k+1}$ selon U. Selon la Définition 5.1, soit T' contient une tilde $(i_t + k, f_t + k)$ qui inclut ou chevauche $(i_b + k, f_b + k)$ et par construction U contient une tilde (i_t, f_t) chevauchant ou incluant (i_b, f_b), soit $v'_{i_b+k} \cdots w'_{v_b+k} \neq \varepsilon$ et ainsi $w'_{i_b+k} \cdots w'_{f_b+k} \neq \varepsilon$. Par conséquent, w' vérifie toute barre de $B_{\geq k+1}$ selon U. Selon la Définition 5.2, w' est dans $L(\overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}))$. Ainsi, puisque w'_k est dans $a^{-1}(L(E_k))$, w est dans $a^{-1(}L(E_k)) \cdot L(\overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}))$. De plus, si $k \neq 1$, le couple $(1, k-1)$ est dans T. Finalement, selon la Définition 6.4, w est dans L'. □

Définition 6.4 *Soit* $E = \overline{\sim}_{T;B}(E_{1,n})$ *une ERA totale sur un alphabet* Σ *et* a *dans* Σ. *Alors :*

$$\frac{d}{d_a}(E) = \begin{pmatrix} \frac{d}{d_a}(E_1) \cdot \overline{\sim}_{T_{\geq 2};B_{\geq 2}}(E_{2,n}) \\ +\sum_{(1,k-1)\in T} \frac{d}{d_a}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}), \end{pmatrix}$$

$$\frac{\partial}{\partial_a}(E) = \begin{cases} \frac{\partial}{\partial_a}(E_1) \cdot \overline{\sim}_{T_{\geq 2};B_{\geq 2}}(E_{2,n}) \\ \cup\bigcup_{(1,k-1)\in T} \frac{\partial}{\partial_a}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}). \end{cases}$$

Nous considérons l'extension usuelle de la dérivation (partielle ou non) par un mot w, à savoir :

$$\frac{d}{d_w}(E) = \begin{cases} E & \text{si } w = \varepsilon, \\ \frac{d}{d_a}(\frac{d}{d_{w'}}(E)) & \text{si } w = w' \cdot a \in \Sigma^+, \end{cases}$$

$$\frac{\partial}{\partial_w}(E) = \begin{cases} E & \text{si } w = \varepsilon, \\ \frac{\partial}{\partial_a}(\frac{\partial}{\partial_{w'}}(E)) & \text{si } w = w' \cdot a \in \Sigma^+. \end{cases}$$

Nous montrons alors les propriétés liant les dérivées et les dérivées partielles puis nous montrons le lien avec le quotient du langage dénoté par une expression dérivée.

Proposition 6.6 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA totale sur un alphabet Σ et w un mot de Σ^*. Alors :*

$$\frac{d}{d_w}(E) \sim_{ACI} \sum_{E' \in \frac{\partial}{\partial_w}(E)} E'.$$

Démonstration. Par récurrence sur la longueur de w et par induction sur la structure de l'expression. Selon le Théorème 2.6, pour toute expression rationnelle simple E, $\frac{d}{d_w}(E) \sim_{ACI} \sum_{E' \in \frac{\partial}{\partial_w}(E)} E'$. Nous étendons cette propriété sur les ERA totales.

Si $w = \varepsilon$, on a $\frac{d}{d_w}(E) = \frac{\partial}{\partial_w}(E) = E$.

Supposons $w = a$. Par hypothèse d'induction, pour tout k de $[\![1, n]\!]$, $\frac{d}{d_a}(E_k) \sim_{ACI} \sum_{E' \in \frac{\partial}{\partial_a}(E_k)} E'$. Par conséquent, selon les formules de la Définition 6.4,

$$\frac{d}{d_a}(E) \sim_{ACI} \sum_{E' \in \frac{\partial}{\partial_a}(E)} E'.$$

Considérons $w = w' \cdot a$ dans Σ^+. Alors :

$$\frac{d}{d_w}(E) = \frac{d}{d_{w'a}}(E) = \frac{d}{d_a}(\frac{d}{d_{w'}}(E)).$$

Par hypothèse de récurrence,

$$\frac{d}{d_{w'}}(E) \sim_{ACI} \sum_{E' \in \frac{\partial}{\partial_{w'}}(E)} E'.$$

Ainsi,

$$\frac{d}{d_a}(\frac{d}{d_{w'}}(E)) \sim_{ACI} \frac{d}{d_a}(\sum_{E' \in \frac{\partial}{\partial_{w'}}(E)} E') = \sum_{E' \in \frac{\partial}{\partial_{w'}}(E)} \frac{d}{d_a}(E').$$

Par hypothèse d'induction, en appliquant le cas de base pour un mot de longueur 1,

$$\frac{d}{d_a}(E') \sim_{ACI} \sum_{E'' \in \frac{\partial}{\partial_a}(E')} E''.$$

Par conséquent,

$$\frac{d}{d_{w'a}}(E) \sim_{ACI} \sum_{E' \in \frac{\partial}{\partial_{w'}}(E), E'' \in \frac{\partial}{\partial_a}(E')} E'' = \sum_{E' \in \frac{\partial}{\partial_{w'a}}(E)} E'.$$

□

Corollaire 6.1 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA totale sur un alphabet Σ et w un mot de Σ^*. Alors :*

$$L(\frac{d}{d_w}(E)) = \bigcup_{E' \in \frac{\partial}{\partial_w}(E)} L(E').$$

Proposition 6.7 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA totale sur un alphabet Σ et w un mot de Σ^*.*

Alors :

$$L(\tfrac{d}{d_w}(E)) = w^{-1}(L(E)).$$

Démonstration. Par récurrence sur la longueur de w. Selon la Proposition 2.1, pour toute expression rationnelle simple E, $L(\frac{d}{d_w}(E)) = w^{-1}(L(E))$. Nous étendons cette propriété sur les ERA totales.

Si $w = \varepsilon$, $L(\frac{d}{d_w}(E)) = L(E) = w^{-1}(L(E))$.

Supposons $w = a$ dans Σ. Par hypothèse d'induction, pour tout entier k de $[\![1, n]\!]$, $a^{-1}(L(E_k)) = L(\frac{d}{d_w}(E_k))$. Selon les formules de la Proposition 6.5 et de la Définition 6.4, $a^{-1}(L(E)) = L(\frac{d}{d_a}(E))$.

Supposons que $w = w' \cdot a$ est dans Σ^+. Alors :

$$L(\tfrac{d}{d_w}(E)) = L(\tfrac{d}{d_{w'a}}(E)) = L(\tfrac{d}{d_a}(\tfrac{d}{d_{w'}}(E))).$$

Selon le cas de base pour $w = a$, pour toute expression F,

$$a^{-1}(L(F)) = L(\tfrac{d}{d_a}(F)).$$

Par conséquent :

$$L(\tfrac{d}{d_a}(\tfrac{d}{d_{w'}}(E))) = a^{-1}(L(\tfrac{d}{d_{w'}}(E))).$$

Par hypothèse de récurrence,

$$w'^{-1}(L(E)) = L(\tfrac{d}{d_{w'}}(E)).$$

Par conséquent,

$$a^{-1}(L(\tfrac{d}{d_{w'}}(E))) = a^{-1}(w'^{-1}L(E)) = (w'a)^{-1}L(E) = w^{-1}L(E).$$

□

Nous démontrons enfin que les ensembles de dérivées et de dérivées partielles sont des ensembles finis, et nous présentons un exemple de construction. Le lemme suivant permet de majorer le cardinal de l'ensembles des dérivées partielles d'une expression par rapport aux mots de Σ^+.

Lemme 6.4 *Soit $E = \overline{\frown}_{T;B}(E_{1,n})$ une ERA totale sur un alphabet Σ et w dans Σ^+. Alors :*

$$\tfrac{\partial}{\partial_w}(E) \subset \bigcup_{w=uv \wedge v \neq \varepsilon} \bigcup_{k=1}^{n} \tfrac{\partial}{\partial_v}(E_k) \cdot \overline{\frown}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}).$$

Démonstration. Par récurrence sur la longueur de w.

Soit a une lettre de Σ. Selon la Définition 6.4,

$$\tfrac{\partial}{\partial_a}(E) = \begin{cases} \tfrac{\partial}{\partial_a}(E_1) \cdot \overline{\frown}_{T_{\geq 2};B_{\geq 2}}(E_{2,n}) \\ \cup \bigcup_{(1,k-1)\in T} \tfrac{\partial}{\partial_a}(E_k) \cdot \overline{\frown}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}). \end{cases}$$

La propriété est ainsi vérifiée pour les mots de longueur 1.

Supposons que $w = w' \cdot a$ est dans Σ^+. Si $w' = \varepsilon$, le cas a déjà été traité. Sinon :

$$\tfrac{\partial}{\partial_w}(E) = \tfrac{\partial}{\partial_{w'a}}(E) = \tfrac{\partial}{\partial_a}(\tfrac{\partial}{\partial_{w'}}(E)).$$

Pour tout k de $[\![1, n]\!]$, posons $R_k = \overline{\frown}_{T_{\geq k};B_{\geq k}}(E_{k,n})$.

Par hypothèse de récurrence,

$$\tfrac{\partial}{\partial_{w'}}(E) \subset \bigcup_{w'=u'v' \wedge v' \neq \varepsilon} \bigcup_{k=1}^{n} \tfrac{\partial}{\partial_{v'}}(E_k) \cdot R_{k+1}.$$

Trivialement, $\frac{\partial}{\partial_{w'}}(E) \subset A$ avec

$$A = \bigcup_{w'=u'v'} \bigcup_{k=1}^{n} \frac{\partial}{\partial_{v'}}(E_k) \cdot R_{k+1}.$$

Ainsi

$$\frac{\partial}{\partial_a}(\frac{\partial}{\partial_{w'}}(E)) \subset \frac{\partial}{\partial_a}(A).$$

De plus,

$$\frac{\partial}{\partial_a}(A) = \bigcup_{w'=u'v'} \bigcup_{k=1}^{n} \frac{\partial}{\partial_a}(\frac{\partial}{\partial_{v'}}(E_k) \cdot R_{k+1}) \subset B$$

avec

$$B = \bigcup_{w'=u'v'} \bigcup_{k=1}^{n} \frac{\partial}{\partial_{v'a}}(E_k) \cdot R_{k+1} \cup \frac{\partial}{\partial_a}(R_{k+1}).$$

Par hypothèse de récurrence,

$$\frac{\partial}{\partial_a}(R_{k+1}) \subset \bigcup_{k'=k+1}^{n} \frac{\partial}{\partial_a}(E_{k'}) \cdot R_{k'}.$$

De plus,

$$\bigcup_{k'=k+1}^{n} \frac{\partial}{\partial_a}(E_{k'}) \cdot R_{k'} \subset \bigcup_{k'=1}^{n} \frac{\partial}{\partial_a}(E_{k'}) \cdot R_{k'}.$$

Par conséquent,

$$\begin{aligned} B &\subset \bigcup_{w'=u'v'} \bigcup_{k=1}^{n} \frac{\partial}{\partial_{v'a}}(E_k) \cdot R_{k+1} \cup \frac{\partial}{\partial_a}(E_k) \cdot R_{k+1} \\ &= \bigcup_{w'a=u'v'a} \bigcup_{k=1}^{n} \frac{\partial}{\partial_{v'a}}(E_k) \cdot R_{k+1} \\ &\subset \bigcup_{w=uv \wedge v \neq \varepsilon} \bigcup_{k=1}^{n} \frac{\partial}{\partial_v}(E_k) \cdot \overline{\sim}_{T_{\geq k+1}; B_{\geq k+1}}(E_{k+1,n}). \end{aligned}$$

□

Proposition 6.8 *Soit E une ERA totale et $\mathcal{D}'_E$ l'ensemble des dérivées partielles de E. Alors :*

$$\#\mathcal{D}'_E \leq |E| + 1.$$

Démonstration. Par induction sur la structure de E et sur sa largeur. Posons $n = |E|$. Considérons $\mathcal{D}^+_E$ l'ensemble des dérivées partielles de E par rapport aux mots de Σ^+. Selon le théorème 3.4 de [2], pour toute expression rationnelle simple F, $\#\mathcal{D}^+_F \leq |F|$.

Posons $E = \overline{\sim}_{T;B}(E_{1,n})$ et soit Σ son alphabet. Pour tout k de $[\![1, n]\!]$, posons $R_k = \overline{\sim}_{T_{\geq k}; B_{\geq k}}(E_{k,n})$. Selon le Lemme 6.4,

$$\begin{aligned} \mathcal{D}^+_E &= \bigcup_{w \in \Sigma^+} \frac{\partial}{\partial_w}(E) \\ &\subset \bigcup_{w \in \Sigma^+} \bigcup_{w=uv} \bigcup_{k=1}^{n} \frac{\partial}{\partial_v}(E_k) \cdot R_{k+1} \\ &= \bigcup_{v \in \Sigma^+} \bigcup_{k=1}^{n} \frac{\partial}{\partial_v}(E_k) \cdot R_{k+1}. \end{aligned}$$

Par conséquent,

$$\begin{aligned} \#\mathcal{D}^+_E &\leq \#\bigcup_{v \in \Sigma^+} \bigcup_{k=1}^{n} \frac{\partial}{\partial_v}(E_k) \cdot R_{k+1} \\ &\leq \#\bigcup_{v \in \Sigma^+} \bigcup_{k=1}^{n} \frac{\partial}{\partial_v}(E_k) \\ &\leq \sum_{k=1}^{n} \#\mathcal{D}^+_{E_k} \\ &\leq \sum_{k=1}^{n} |E_k| = |E| = n. \end{aligned}$$

Ainsi, puisque

$$\#\mathcal{D}'_E = \#\mathcal{D}^+_E \cup \frac{\partial}{\partial_\varepsilon}(E),$$

on a

$$\#\mathcal{D}'_E \leq n + 1.$$

□

Proposition 6.9 *Soit E une ERA totale et $\mathcal{D}_E$ l'ensemble des dérivées de E. Alors :*
*$\mathcal{D}_E$ est un ensemble fini modulo **ACI**.*

Démonstration. Considérons Σ l'alphabet de E. Soit $\mathcal{D}'_E$ l'ensemble des dérivées partielles de E. Selon la Proposition 6.8, $\#\mathcal{D}'_E \leq n+1$. Selon la Proposition 6.6, pour tout w de Σ, $\frac{d}{d_w}(E) \sim_{ACI} \sum_{E' \in \frac{\partial}{\partial w}(E)} E'$. Par conséquent, il existe au plus 2^{n+1} dérivées différentes modulo ACI, et $\mathcal{D}_E$ est un ensemble fini modulo **ACI**. □

Exemple 6.8 *Considérons l'ERA en FoTo* $E = \overline{\sim}_{(1,1),(2,2);(1,2)}\big((a+b),(a+b)\big)\cdot a^*$ *représentée par* $\overline{(\widetilde{a+b})(\widetilde{a+b})}\cdot a^*$. *Les dérivées et les dérivées partielles de E sont les expressions suivantes :*

$$
\begin{aligned}
\tfrac{d}{d_\varepsilon}(E) &= E,\\
\tfrac{d}{d_a}(E) &= \tfrac{d}{d_a}(\overline{\sim}_{(1,1),(2,2);(1,2)}\big((a+b),(a+b)\big))\cdot a^*\\
&= \tfrac{d}{d_a}(\overline{(\widetilde{a+b})(\widetilde{a+b})})\cdot a^*,\\
&= \tfrac{d}{d_a}(a+b)\cdot \overline{\sim}_{(1,1);\emptyset}\big(a+b\big)\cdot a^* + \tfrac{d}{d_a}(a+b)\cdot a^*\\
&= \tfrac{d}{d_a}(a+b)\cdot \widetilde{a+b}\cdot a^* + \tfrac{d}{d_a}(a+b)\cdot a^*,\\
&= \overline{\sim}_{(1,1);\emptyset}\big(a+b\big)\cdot a^* + a^*\\
&= \widetilde{a+b}\cdot a^* + a^*,\\
\tfrac{d}{d_b}(E) &= \tfrac{d}{d_b}(\overline{(\widetilde{a+b})(\widetilde{a+b})})\cdot a^*,\\
&= \tfrac{d}{d_b}(a+b)\cdot \widetilde{a+b}\cdot a^* + \tfrac{d}{d_b}(a+b)\cdot a^*,\\
&= \widetilde{a+b}\cdot a^* + a^*,\\
\tfrac{d}{d_a}(\widetilde{a+b}\cdot a^* + a^*) &= \tfrac{d}{d_a}(\widetilde{a+b}\cdot a^*) + \tfrac{d}{d_a}(a^*)\\
&= \tfrac{d}{d_a}(\widetilde{a+b})\cdot a^* + \tfrac{d}{d_a}(a^*) + a^*,\\
&= \tfrac{d}{d_a}(a+b)\cdot a^* + a^* + a^*,\\
&= a^* + a^* + a^*,\\
&\sim_{ACI} a^*,\\
\tfrac{d}{d_b}(\widetilde{a+b}\cdot a^* + a^*) &= \tfrac{d}{d_b}(\widetilde{a+b}\cdot a^*) + \tfrac{d}{d_b}(a^*)\\
&= \tfrac{d}{d_b}(\widetilde{a+b})\cdot a^* + \tfrac{d}{d_b}(a^*) + \emptyset,\\
&= \tfrac{d}{d_b}(a+b)\cdot a^* + \emptyset,\\
&= a^*,\\
\tfrac{d}{d_a}(a^*) &= a^*,\\
\tfrac{d}{d_b}(a^*) &= \emptyset,
\end{aligned}
$$

$$
\begin{aligned}
\tfrac{\partial}{\partial_\varepsilon}(E) &= \{E\},\\
\tfrac{\partial}{\partial_a}(E) &= \tfrac{\partial}{\partial_a}(\overline{(\widetilde{a+b})(\widetilde{a+b})})\cdot a^*,\\
&= \tfrac{\partial}{\partial_a}(a+b)\cdot \widetilde{a+b}\cdot a^* \cup \tfrac{\partial}{\partial_a}(a+b)\cdot a^*,\\
&= \{\widetilde{a+b}\cdot a^*, a^*\},
\end{aligned}
$$

$$\begin{aligned}
\tfrac{\partial}{\partial_b}(E) &= \tfrac{\partial}{\partial_b}(\overbrace{(\widetilde{a+b})(\widetilde{a+b})}) \cdot a^*,\\
&= \tfrac{\partial}{\partial_b}(a+b)\cdot \widetilde{a+b}\cdot a^* \cup \tfrac{\partial}{\partial_b}(a+b)\cdot a^*,\\
&= \{\widetilde{a+b}\cdot a^*, a^*\},\\
\tfrac{\partial}{\partial_a}(\widetilde{a+b}\cdot a^*) &= \tfrac{\partial}{\partial_a}(\widetilde{a+b})\cdot a^* \cup \tfrac{\partial}{\partial_a}(a^*),\\
&= \{a^*\}\cup\{a^*\},\\
&= \{a^*\},\\
\tfrac{\partial}{\partial_b}(\widetilde{a+b}\cdot a^*) &= \tfrac{\partial}{\partial_b}(\widetilde{a+b})\cdot a^* \cup \tfrac{\partial}{\partial_b}(a^*),\\
&= \{a^*\},\\
\tfrac{\partial}{\partial_a}(a^*) &= \{a^*\},\\
\tfrac{\partial}{\partial_b}(a^*) &= \emptyset.
\end{aligned}$$

L'ensemble des dérivées de E est l'ensemble $\mathcal{D}_E = \{E, \widetilde{a+b}\cdot a^* + a^*, a^*\}$.

L'ensemble des dérivées partielles de E est l'ensemble $\mathcal{D}'_E = \{E, \widetilde{a+b}\cdot a^*, a^*\}$.
L'automate des dérivées de E et l'automate des dérivées partielles de E sont représentés Figure 6.11 et Figure 6.12.

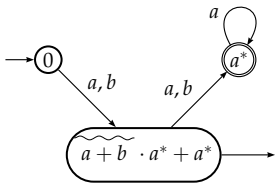


FIGURE 6.11 – *L'Automate des Dérivées de E.*

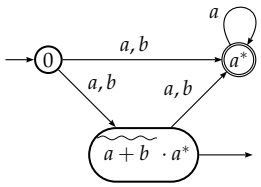


FIGURE 6.12 – *L'Automate des Dérivées Partielles de E.*

6.5 C-Continuations d'une ERA

Dans cette section, nous présentons une extension des formules de c-dérivation de la Définition 2.9 et nous définissons les c-continuations d'une ERA E. Nous commençons par établir la formule de c-dérivation d'une ERA en multi-tildes-barres, puis nous montrons que dans le cas d'une expression linéaire E sur un alphabet Σ, le langage dénoté par la c-dérivée de E par un mot de Σ^* est égal au quotient de $L(E)$ par w.

Définition 6.5 *Soit* $E = \overline{\sim}_{T;B}(E_{1,n})$ *une ERA totale sur un alphabet* Σ *et* a *dans* Σ. *Soit* $K = \{k \in [\![1,n]\!] \mid \frac{d}{d_a}(E_k) \neq \emptyset \wedge (k = 1 \vee (1, k-1) \in T)\}$. *Alors*

$$d_a(E) = \begin{cases} \emptyset & \text{si } K = \emptyset, \\ d_a(E_k)\cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}) & \text{avec } k = \mathrm{Min}(K) \text{ sinon.} \end{cases}$$

Nous considérons l'extension usuelle de la c-dérivation par un mot w :

$$d_w(E) = \begin{cases} E & \text{si } w = \varepsilon, \\ d_a(d_{w'}(E)) & \text{si } w = w'\cdot a \in \Sigma^+. \end{cases}$$

Proposition 6.10 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA totale $\emptyset$-linéaire sur un alphabet Σ et w un mot de Σ^*. Alors :*

$$L(d_w(E)) = w^{-1}(L(E)).$$

Démonstration. Par récurrence sur la longueur de w et par induction sur la structure de E. Selon la Proposition 2.2, pour toute expression rationnelle simple, $L(d_w(E)) = w^{-1}(L(E))$. Nous étendons cette propriété aux ERA totales.

Considérons $w = \varepsilon$. Ainsi, $L(d_w(E)) = L(E) = w^{-1}(L(E))$.

Supposons $w = a$. Par hypothèse d'induction, pour tout k de $[\![1, n]\!]$, $L(d_a(E_k)) = a^{-1}(L(E_k))$. Selon les formules de la Proposition 6.5 et de la Définition 6.5, puisque E est $\emptyset$-linéaire, on a $L(d_a(E)) = a^{-1}(L(E))$.

Considérons un mot $w = w'a$. Alors on a :

$$w^{-1}(L(E)) = a^{-1}(w'^{-1}(L(E))).$$

Par hypothèse de récurrence,

$$w'^{-1}(L(E)) = L(d_{w'}(E)).$$

Ainsi,

$$w^{-1}(L(E)) = a^{-1}(L(d_{w'}(E))).$$

Par hypothèse d'induction sur le cas de base de longueur 1,

$$a^{-1}(L(d_{w'}(E))) = L(d_a(d_{w'}(E))) = L(d_{aw'}(E)) = L(d_w(E)).$$

□

Nous montrons alors comment calculer la c-dérivée d'une ERA par rapport à un mot w et nous démontrons que si cette dérivée est non-nulle, alors sa valeur ne dépend que de la dernière lettre de w.

Proposition 6.11 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA totale $\emptyset$-linéaire sur un alphabet Σ, Σ_l l'alphabet de E_l pour tout l de $[\![1, n]\!]$. Soit $w = w' \cdot w''$ un mot de Σ^+ et k un entier tel que w'' est le plus long suffixe de w appartenant à Σ_k^*. Alors :*

$$d_w(E) = \begin{cases} d_{w''}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}) & \text{si } \varepsilon \in L(d_{w'}(\overline{\sim}_{T_{\leq k-1};B_{\leq k-1}}(E_{1,k-1}))), \\ \emptyset & \text{sinon.} \end{cases}$$

Démonstration. Par récurrence sur la longueur de w et sur la structure de E. Le cas de base de la récurrence est le cas où $w = a$. Selon la Définition 6.5, la formule est vérifiée.

Supposons que la propriété soit vraie pour un mot $w = w' \cdot w''$.

Si $d_w(E) = \emptyset$, $d_{wa}(E) = d_a(\emptyset) = \emptyset$.

Supposons maintenant que $d_w(E) \neq \emptyset$. Ainsi,

$$d_w(E) = d_{w''}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n})$$
$$\text{et } \varepsilon \in L(d_{w'} \overline{\sim}_{T_{\leq k-1};B_{\leq k-1}}(E_{1,k-1})).$$

Deux cas sont à traiter :

(a) Supposons que a soit dans Σ_k. Alors :

$$\begin{aligned} d_{wa}(E) &= d_a(d_{w''}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n})) \\ &= d_a(d_{w''}(E_k)) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}) \\ &= d_{w''a}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}). \end{aligned}$$

Puisque $w''a$ est le plus long suffixe de $w \cdot a$ dans Σ_k^* et que w' n'a pas changé, la formule est vérifiée.

(b) Supposons que a est dans $\Sigma_{k'}$ avec $k' \geq k+1$. Alors, si $\varepsilon \notin L(d_{w''}(E_k))$,

$$d_{wa}(E) = d_a(d_{w''}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n})) = \emptyset.$$

Sinon $\varepsilon \in L(d_{w''}(E_k))$ et

$$\begin{aligned} d_{wa}(E) &= d_a(d_{w''}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n})) \\ &= d_a(\overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n})) \end{aligned}$$

Si $k' = k+1$, on a

$$d_{wa}(E) = d_a(E_{k+1}) \cdot \overline{\sim}_{(T_{\geq k+1})_{\geq 2};(B_{\geq k+1})_{\geq 2}}(E_{k+2,n}).$$

Si $k' > k+1$, selon la Définition 6.5, soit $(1, k'-k-1) \notin T_{\geq k+1}$ et alors

$$d_a(\overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n})) = \emptyset,$$

soit $(1, k'-k-1) \in T_{\geq k+1}$ et

$d_a(\overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n})) = d_a(E_{k'}) \cdot \overline{\sim}_{(T_{\geq k+1})_{\geq k'-k+1};(B_{\geq k'-k+1})_{\geq k'-k+1}}(E_{k'+1,n})$.

Le mot a est le plus long suffixe de w appartenant à $\Sigma_{k'}^*$. Montrons alors que ε est dans $L(d_{w'w''} \overline{\sim}_{T_{\leq k'-1};B_{\leq k'-1}}(E_{1,k}))$.

Par hypothèse d'induction, ε appartient à $L(d_{w'} \overline{\sim}_{T_{\leq k-1};B_{\leq k-1}}(E_{1,k-1}))$; par conséquent, w' appartient à $L(\overline{\sim}_{T_{\leq k-1};B_{\leq k-1}}(E_{1,k-1}))$. Puisque ε est dans $L(d_{w''}(E_k))$, alors w'' appartient à $L(E_k)$. Ainsi, selon la Définition 5.2, il existe une sous-liste libre T' de $T_{\leq k-1}$ générant w' et vérifiant toutes les barres de $B_{\leq k-1}$. Puisque w'' est dans Σ^+, alors T' est une sous-liste libre de $T_{\leq k}$ génèrant $w' \cdot w''$ et vérifiant toutes les barres de $B_{\leq k-1}$. Puisque w'' est dans Σ^+, toute barre se finissant en k est vérifiée par $w' \cdot w''$ selon T'. Par conséquent $w' \cdot w''$ vérifie toutes les barres de $B_{\leq k}$ selon T'. Selon la Définition 5.2, $w = w' \cdot w''$ est dans $L(\overline{\sim}_{T_{\leq k};B_{\leq k}}(E_{1,k})))$. Et par conséquent, ε appartient à $L(d_{w'w''} \overline{\sim}_{T_{\leq k};B_{\leq k}}(E_{1,k}))$. Si $k' = k+1$, la formule est alors vérifiée. Si $k' > k+1$, le couple $(1, k'-k-1)$ est dans $T_{\geq k+1}$, et par définition $(k+1, k'-1)$ appartient à T ; par conséquent $(k+1, k'-1)$ est dans $T_{\leq k'-1}$. La liste $T' \cup (k+1, k'-1)$ est ainsi une sous-liste libre de $T_{\leq k'-1}$ générant $w' \cdot w''$ et vérifiant toutes les barres de $B_{\leq k+1}$. Puisque toute barre se finissant à un rang compris entre $k+1$ et $k'-1$ soit chevauche le couple $(k+1, k'-1)$, soit est incluse dans le couple $(k+1, k'-1)$, la liste $T' \cup (k+1, k'-1)$ est une sous-liste libre de $T_{\leq k'-1}$ générant $w' \cdot w''$ et vérifiant toutes les barres de $B_{\leq k'-1}$. Selon la Définition 5.2, $w = w' \cdot w''$ est dans $L(\overline{\sim}_{T_{\leq k'-1};B_{\leq k'-1}}(E_{1,k'-1})))$. Et par conséquent, ε est dans $L(d_{w'w''} \overline{\sim}_{T_{\leq k'-1};B_{\leq k'-1}}(E_{1,k'-1}))$. La formule est alors vérifiée. □

Théorème 6.1 *Soit E une ERA totale linéaire sur un alphabet Σ. Soit a un symbole de Σ. Pour tout mot u, v de Σ^*, la proposition suivante est vérifiée :*

$$(d_{ua}(E) \neq \emptyset \wedge d_{va}(E) \neq \emptyset) \Rightarrow d_{ua}(E) = d_{va}(E).$$

Démonstration. Par induction sur la structure de E. Selon le Théorème 2.3, pour toute expression rationnelle simple F,

$$(d_{ua}(F) \neq \emptyset \wedge d_{va}(F) \neq \emptyset) \Rightarrow d_{ua}(F) = d_{va}(F).$$

Nous étendons cette propriété sur les ERA totales.

Supposons que $d_{ua}(E) \neq \emptyset$ et que $d_{va}(E) \neq \emptyset$. Selon la Proposition 6.11, il existe k dans $[\![1, n]\!]$ tel que $ua = u'u''a$ et $va = v'v''a$ avec $u''a$ le plus long suffixe de ua dans Σ_k^+ et $v''a$ le plus long suffixe de va dans Σ_k^+. Alors

$$d_{ua}(E) = d_{u''a}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}).$$

Par hypothèse d'induction,

$$d_{u''a}(E_k) = d_{v''a}(E_k).$$

Ainsi,

$$\begin{aligned} d_{ua}(E) &= d_{u''a}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}) \\ &= d_{v''a}(E_k) \cdot \overline{\sim}_{T_{\geq k+1};B_{\geq k+1}}(E_{k+1,n}) \\ &= d_{va}(E). \end{aligned}$$

□

Cette unicité de la c-dérivée d'une ERA E par rapport à un mot wa nous permet d'étendre la notion de c-continuation d'une expression par rapport à une lettre aux ERA.

Proposition 6.12 *Soit $E = \overline{\sim}_{T;B}(E_{1,n})$ une ERA totale $\emptyset$-linéaire sur un alphabet Σ avec Σ_k l'alphabet de E_k pour tout k de $[\![1, n]\!]$. Soit x un symbole de Σ_l. Alors :*

$$c_x(E) = c_x(E_l) \cdot \overline{\sim}_{T_{\geq l+1};B_{\geq l+1}}(E_{l+1,n}).$$

Démonstration. Par induction sur la structure de E. Comme $x \in \Sigma$, il existe un mot w de Σ^* tel que $d_{wx}(E) \neq \emptyset$ et donc $c_x(E) = d_{wx}(E)$. Considérons $w = w' \cdot w''$ avec w'' plus long suffixe de w dans Σ_l^*. Ainsi $w''x$ est le plus long suffixe de wx dans Σ_l^+. Selon la Proposition 6.11,

$$d_{wx}(E) = d_{w''x}(E_l) \cdot \overline{\sim}_{T_{\geq l+1};B_{\geq l+1}}(E_{l+1,n}).$$

Puisque $d_{wx}(E) \neq \emptyset$, alors $d_{w''x}(E_l) \neq \emptyset$. Par induction sur la structure de E, on a $d_{w''x}(E_l) = c_x(E_l)$. Finalement,

$$c_x(E) = c_x(E_l) \cdot \overline{\sim}_{T_{\geq l+1};B_{\geq l+1}}(E_{l+1,n}).$$

□

Exemple 6.9 *Considérons l'ERA en FoTo $E = \overline{\sim}_{(1,1),(2,2);(1,2)}\big((a+b),(a+b)\big) \cdot a^*$ représentée par $\overline{\widetilde{(\ a+b\ )}\widetilde{(\ a+b\ )}} \cdot a^*$. Soit $F = \overline{\sim}_{(1,1),(2,2);(1,2)}\big((1+2),(3+4)\big) \cdot 5^*$ l'expression $\emptyset$-linéarisée de E.*

$$
\begin{aligned}
c_1(F) &= c_1(1+2)\cdot \widetilde{\ }_{(1,1);\varnothing}(3+4)\cdot 5^* \\
&= \widetilde{c_1(1)\cdot\ 3+4\ }\cdot 5^* \\
&= \widetilde{\ 3+4\ }\cdot 5^* \\
c_2(F) &= c_2(1+2)\cdot \widetilde{\ }_{(1,1);\varnothing}(3+4)\cdot 5^* \\
&= \widetilde{c_2(2)\cdot\ 3+4\ }\cdot 5^* \\
&= \widetilde{\ 3+4\ }\cdot 5^* \\
c_3(F) &= c_3(3+4)\cdot 5^* \\
&= c_3(3)\cdot 5^* \\
&= 5^* \\
c_4(F) &= c_4(3+4)\cdot 5^* \\
&= c_4(4)\cdot 5^* \\
&= 5^* \\
c_5(F) &= c_5(5^*) \\
&= c_5(5)\cdot 5^* \\
&= 5^*
\end{aligned}
$$

Les c-continuations de F sont

$$\mathcal{C}_F = \{c_1(F) = c_2(F) = \widetilde{\ 3+4\ }\cdot 5^*, c_3(F) = c_4(F) = c_5(F) = 5^*\}.$$

L'automate des c-continuations de F est représenté Figure 6.13.

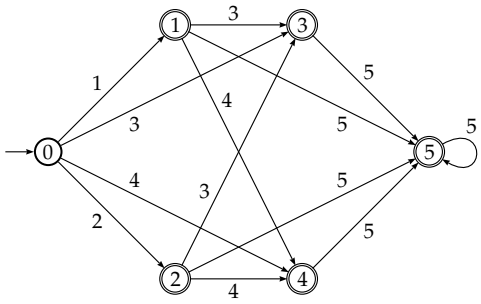


FIGURE 6.13 – *L'Automate des c-Continuations de F.*

6.6 D'un NFA de Glushkov Étendu vers une ERA

Dans cette section, nous montrons qu'il existe une conversion efficace en ERA pour les automates de Glushkov étendus, définis comme suit.

Définition 6.6 *Soit A un automate. L'automate A est un* automate de Glushkov étendu *s'il existe une ERA E telle que l'automate des positions de E soit isomorphe à A.*

Nous montrons tout d'abord que tout automate à $(n+1)$ états standard, homogène et en ligne peut être transformé en une ERA de largeur n dans la Sous-Section 6.6.1. Nous étendons ensuite ce résultat aux automates acycliques dans la Sous-Section 6.6.2. Enfin, la Sous-Section 6.6.3

étend l'algorithme CZ et permet ainsi de caractériser les automates de Glushkov étendus. Dans la suite de cette section, nous considérons des automates émondés.

6.6.1 D'un NFA Standard, Homogène et en Ligne vers une ERA

Tout automate A à $(n+1)$ états standard, homogène et en ligne peut être converti en une ERA de largeur n. Nous montrons tout d'abord comment calculer une ERA E' dénotant le langage reconnu par l'automate jumeau A' de A, puis nous appliquons une délinéarisation pour obtenir une ERA E de largeur n dénotant $L(A)$.

Définition 6.7 *Soit $A' = (\Sigma', Q, \{0\}, F, \delta')$ l'automate jumeau d'un NFA A. Soit $n = \#Q - 1$. Considérons les suites T' et B' définies par :*

$$T' = ((i,f) \mid f+1 \in \delta'(i-1, f+1)) \cup ((i,n) \mid i-1 \in F),$$
$$B' = [\![1,n]\!]^2_{\leq} \setminus T'.$$

*L'expression $E' = \widetilde{\ }_{T';B'}(1,\ldots,n)$ est l'*expression caractéristique *de A'.*

Exemple 6.10 *Considérons l'automate jumeau de la Figure 6.2. Son expression caractéristique est l'ERA $\widetilde{\ }_{T';B'}(1,2,3)$ avec $T' = ((1,2),(2,3))$ et $B' = [\![1,3]\!]^2_{\leq} \setminus T'$.*

L'expression caractéristique E' d'un automate jumeau A' est clairement totale, plate, et $\emptyset$-linéaire. Elle est ainsi en FoAm. Nous étudions maintenant le langage de l'expression caractéristique E' de A'. Remarquons que l'application h_A qui est définie lors de la construction de A' depuis A peut être vue comme un morphisme alphabétique de Σ'^* vers Σ^*. Nous dirons que h_A est le *morphisme associé à A*. Ce morphisme permet de transformer l'expression E' en une expression E dénotant le langage reconnu par A. L'expression E est l'expression canonique de A et reconnait $L(A)$.

Lemme 6.5 *Soit A un automate standard, homogène et en ligne et A' l'automate jumeau de A. Soit h_A le morphisme associé à A. Alors :*

$$\mathrm{h}_A(L(A')) = L(A).$$

Démonstration. Par construction de A' depuis A, pour tout w de Σ^*, w est dans $L(A)$ si et seulement s'il existe w' dans $L(A')$ tel que $\mathrm{h}_A(w') = w$. □

Proposition 6.13 *Soit A' l'automate jumeau d'un NFA A à $(n+1)$ états. Soit E' l'expression caractéristique de A'. L'ERA E' vérifie les propriétés suivantes :*

(1) *E' est de largeur n,*

(2) *$L(A') = L(E')$,*

(3) *E' peut être calculée en temps $O(n^2)$ depuis A'.*

Démonstration. Posons $E' = \widetilde{\ }_{T';B'}(1,\ldots,n)$.

(1) Puisque E' est plate, $|E'| = n$.

(2a) Soit w un mot de $L(E')$. Selon la Définition 5.2, w se décompose en $w_1 \cdots w_n$. Selon le Lemme 5.8, pour tout facteur $w_i \cdots w_f$ ε-maximal de w, (i, f) est dans T. Par construction, soit $\delta'(i-1, f+1) = f+1$ si $f \neq n$, soit $(i-1) \in F$. Puisque l'automate est en ligne, pour tout $k \in [\![0, n-1]\!]$, $\delta'(k, k+1) = k+1$. Par conséquent, w est l'étiquette d'un chemin réussi de A', et ainsi $w \in L(A')$.

(2b) Soit w un mot de $L(A')$. Puisque A' est en ligne, w se décompose en $w_1 \cdots w_n$ tel que pour tout k de $[\![1, n]\!]$, soit $w_k = k$, soit $w_k = \varepsilon$. Puisque l'automate est en ligne, pour tout $k \in [\![0, n-1]\!]$, $\delta'(k, k+1) = k+1$. Pour tout facteur $w_i \cdots w_f$ ε-maximal de w, soit $\delta'(i-1, f+1) = f+1$ si $f \neq n$, soit $(i-1) \in F$. Par construction, (i, f) est dans T. Selon le Lemme 5.8, w est dans $L(E')$.

(3) La construction des listes T et B nécessite un temps quadratique correspondant aux nombres de transitions de A. Par conséquent, la construction se réalise en temps $O(n^2)$. □

Définition 6.8 *Soit A un automate à $(n+1)$ états standard, homogène et en ligne, A' l'automate jumeau de A et $E' = \overline{\sim}_{T';B'}(1, \ldots, n)$ l'expression caractéristique A'. L'ERA $E = \overline{\sim}_{T';B'}(\mathrm{h}_A(1), \ldots, \mathrm{h}_A(n))$ est l'*expression canonique de A.

Proposition 6.14 *Soit A un ASHE à $(n+1)$ états, A' l'automate jumeau de A, E' l'expression caractéristique de A' et E l'expression canonique de A. L'ERA E vérifie les propriétés suivantes :*

(1) *$|E| = n$,*

(2) *$L(E) = L(A)$,*

(3) *E est calculée en temps $O(n)$ depuis E'.*

Démonstration. Posons $A = (\Sigma, Q, I, F, \delta)$, $A' = (\Sigma', Q', I', F', \delta')$ l'automate jumeau de A, $E' = \overline{\sim}_{T;B}(1, \ldots, n)$ l'expression caractéristique de A' et $E = \overline{\sim}_{T;B}(\mathrm{h}_A(1), \ldots, \mathrm{h}_A(n))$ l'expression canonique de E.

(1) Par construction, $|E| = |E'| = \#Q' - 1 = \#Q - 1$.

(2) Remarquons que l'ERA $E' = \overline{\sim}_{T;B}(1, \ldots, n)$ est l'expression linéarisée de $E = \overline{\sim}_{T;B}(\mathrm{h}_A(1), \ldots, \mathrm{h}_A(n))$ et que le morphisme de linéarisation est h_A. De plus, l'expression E' est totale, plate et a une largeur égale à n. Puisque les opérateurs de multi-tildes-barres sont compatibles avec la linéarisation, $L(E) = \mathrm{h}_A(L(E'))$. Selon le Lemme 6.5, $\mathrm{h}_A(L(A')) = L(A)$ et selon la Proposition 6.13, $L(A') = L(E')$. Cela implique que $\mathrm{h}_A(L(E')) = L(A)$ et finalement $L(A) = L(E)$.

(3) Les listes de tildes et de barres de E et de E' sont égales, et il faut délinéariser les n symboles de E'. La construction est ainsi réalisée en temps $O(n)$. □

Exemple 6.11 *Considérons l'automate de la Figure 6.1. L'expression caractéristique de son automate jumeau, donnée dans l'Exemple 6.10, est l'ERA $E' = \overline{\sim}_{T';B'}(1, 2, 3)$*

avec $T' = ((1,2),(2,3))$ *et* $B' = [\![1,3]\!]^2_{\leq} \setminus T'$. *L'expression canonique de A est l'expression* $E = \widetilde{\ }_{T';B'}(a,b,c)$.

Ainsi, tout ASHE est un automate de Glushkov étendu.

Proposition 6.15 *Soit A un ASHE. Alors A est un automate de Glushkov étendu.*

Démonstration. Par définition de l'expression canonique E de A, l'automate des positions de E est isomorphe à A. □

6.6.2 D'un Automate Acyclique vers une ERA

Nous étendons la Proposition 6.15 aux automates acycliques. Dans un premier temps, nous considérons les automates acycliques standards et homogènes. Pour un tel automate A possédant n états, nous construisons un ASHE A' avec au plus $2(n-1)$ états tel que $L(A) \subset L(A)'$.

Définition 6.9 *Soit* $A = (\Sigma, Q, \{q_0\}, F, \delta)$ *un automate acyclique standard et homogène. Soit* h_A *l'application de* Q *vers* Σ *tel que pour tout* q *de* Q, $\mathrm{h}_A(q)$ *est le symbole étiquetant toute transition entrant en* q. *Soit* τ *un tri topologique de A. Soit* $\perp$ *l'ensemble* $\{\perp_k \mid \tau^{-1}(k+1) \notin \delta(\tau^{-1}(k), \mathrm{h}_A(\tau^{-1}(k+1)))\}$. *L'automate* $A' = (\Sigma', Q', I', F', \delta')$, *appelé* automate amplifié *de A* selon τ, *est défini par :*

- $\Sigma' = \Sigma \cup \perp$, $I' = \{q_0\}$, $Q' = Q \cup \perp$, $F' = F$,
- $\forall (p,a) \in Q' \times \Sigma'$, $\delta'(p,a) = \begin{cases} \delta(p,a) & \text{si } (p,a) \in Q \times \Sigma, \\ \tau^{-1}(k+1) & \text{si } p = \perp_k \wedge a = \mathrm{h}_A(\tau^{-1}(k+1)), \\ \perp_k & \text{si } a = \perp_k \wedge \tau(p) = k, \\ \emptyset & \text{sinon.} \end{cases}$

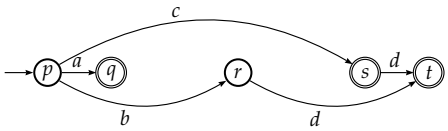


FIGURE 6.14 – *Un Automate Acyclique Standard et Homogène...*

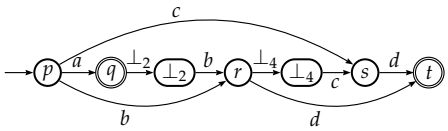


FIGURE 6.15 – *... et un de ses Automates Amplifiés.*

Lemme 6.6 *Soit* $A = (\Sigma, Q, \{q_0\}, F, \delta)$ *un automate acyclique standard et homogène à n états et* $A' = (\Sigma', Q', I', F', \delta')$ *un de ses automates amplifiés. Alors* A' *est un ASHE avec au plus* $2(n-1)$ *états et tel que* $L(A') \cap \Sigma^* = L(A)$.

Démonstration. Par construction, remarquons que :
(a) au plus $n-2$ nouveaux états $\perp_k$ sont créés dans l'automate amplifié,
(b) $\delta = \delta' \cap (Q \times \Sigma \times Q)$; ainsi, pour tout mot w de Σ^*, w est dans $L(A)$ si et seulement si w est dans $L(A')$,
(c) A' est un ASHE. □

Selon la Proposition 6.14, on peut calculer une ERA E' de largeur n' telle que $L(A') = L(E')$. Nous montrons maintenant comment calculer une ERA E de largeur $n-1$ telle que $L(E) = L(A)$.

Proposition 6.16 *Soit A un automate acyclique standard homogène à $(n+1)$ états. Soit A' un automate amplifié de A et $E' = \widetilde{\ }_{T;B}(E'_{1,n'})$ l'ERA canonique de A'. Considérons l'ERA $E = \widetilde{\ }_{T;B}(E_{1,n'})$ telle que pour tout $k \in [\![1, n']\!]$, $E_k = \emptyset$ si $E'_k = \bot_{k'}$, $E_k = E'_k$ sinon. Alors l'ERA E de largeur n peut être calculée en temps $O(n')$ depuis E' et est telle que $L(E) = L(A)$.*

Démonstration. L'expression E est calculée en remplaçant les expressions $E'_k = \bot_{k'}$ par $E_k = \emptyset$ pour $k \in [\![1, n']\!]$. Le test étant de complexité $O(1)$, la construction s'effectue en temps $O(n')$.

Si A est en ligne, alors la Proposition 6.14 s'applique.

Sinon, selon la Définition 6.9, on peut calculer un automate amplifié A' de taille $n' + 1 = n + m + 1$, où m est le nombre de symboles $\bot_k$. Selon le Lemme 6.6, l'automate A' est un ASHE tel que $L(A) = L(A') \cap \Sigma^*$. Selon la Proposition 6.14, on calcule une ERA plate et totale $E' = \widetilde{\ }_{T;B}(E'_{1,n'})$ de largeur n' et telle que $L(E') = L(A')$. Ainsi, $L(A) = L(E') \cap \Sigma^*$. Par construction, l'expression E et l'automate A ont le même alphabet, Σ. Montrons que $L(E) = L(E') \cap \Sigma^*$. **(a)** Soit $w \in L(E)$. Selon le Lemme 5.8, puisque E est une ERA totale, il existe une décomposition $w_1 \cdots w_{n'}$ de w telle que pour tout facteur $w_i \cdots w_f$ ε-maximal, (i, f) est dans T. Remarquons que pour toute expression E_k tel que $E_k = \emptyset$, on a $w_k = \varepsilon$. Puisque E' et E sont définis sur la même liste de couples, selon le Lemme 5.8, $w \in L(E')$. **(b)** Soit w un mot de $L(E') \cap \Sigma^*$. Puisque l'ERA E' est totale, selon le Lemme 5.8, il existe une décomposition $w_1 \cdots w_{n'}$ de w telle que pour tout facteur $w_i \cdots w_f$ ε-maximal de w, (i, f) est dans T. Puisque $w \in \Sigma^*$, pour toute expression E'_k telle que $E'_k = \bot_k$, $w_k = \varepsilon$. Puisque E' et E sont définies sur la même liste de couples, selon le Lemme 5.8, $w \in L(E)$.

Ainsi, pour tout mot $w \in \Sigma^*$,

$$w \in L(A) \Leftrightarrow w \in L(A') \cap \Sigma^* \Leftrightarrow w \in L(E') \cap \Sigma^* \Leftrightarrow w \in L(E).$$

De plus, puisque la largeur de E' est n' et que pendant le calcul de E, les m symboles $\bot_k$ sont remplacés par $\emptyset$, la largeur de E est égale à n. □

Ainsi, tout automate acyclique standard homogène est un automate de Glushkov étendu.

Proposition 6.17 *Soit A un automate acyclique standard homogène. Alors A est un automate de Glushkov étendu.*

Démonstration. Par définition de l'expression canonique E' de l'automate amplifié A' de A, l'automate des positions de E' est isomorphe à A'. Par délinéarisation, on passe de E' à E et de A' à A. L'automate des positions de E est donc isomorphe à A. □

Nous considérons alors la classe des automates acycliques et nous montrons que tout automate acyclique A peut être transformé en une ERA

E de largeur $O(n)$. Remarquons que A n'est pas nécéssairement un automate de Glushkov étendu. Le résultat est basé sur le lemme suivant.

Lemme 6.7 *Soit $A = (\Sigma, Q, I, F, \delta)$ un automate acyclique. Alors un automate équivalent standard et homogène $A' = (\Sigma, Q', I', F', \delta')$, et tel que $\#Q' \leq \#\Sigma \times \#Q + 1$ peut être calculé en temps $O(\#\Sigma \times n^2)$.*

Démonstration. Par application de l'Algorithme 1.7, on obtient à partir d'un NFA A à n états un NFA homogène A' équivalent avec $n' \leq \#\Sigma \times n$ états et $O(\#\Sigma \times n^2)$ transitions. La complexité en temps de cette transformation est $O(\#\Sigma \times n^2)$. Par application de l'Algorithme 1.6, on obtient à partir de A' un NFA A'' standard équivalent avec $n'' \leq n' + 1 \leq \#\Sigma \times n + 1$ états et $O(\#\Sigma \times n^2)$ transitions. La complexité en temps de cette transformation est $O(\#\Sigma \times n^2)$. Remarquons que l'ajout du nouvel état initial non entrant conserve l'homogénéité, et ainsi A'' est également homogène. □

Proposition 6.18 *Soit A un automate acyclique à n état sur un alphabet Σ. Alors on peut construire une ERA E de taille $O(\#\Sigma \times n)$ telle que $L(E) = L(A)$ en temps $O(\#\Sigma \times n^2)$.*

Démonstration. Soit A un automate acyclique à n états sur un alphabet Σ. Selon le Lemme 6.7, un automate standard homogène et équivalent A' peut être calculé, avec moins de $\#\Sigma \times \#Q$ états. Ainsi, selon la Proposition 6.16, A' peut être converti en une ERA de largeur inférieure ou égale à $\#\Sigma \times \#Q$. □

6.6.3 Caractérisation des Automates de Glushkov Étendus

Nous présentons dans cette sous-section une extension du Théorème 2.7 de Caron et Ziadi afin de caractériser les automates de Glushkov étendus. Nous avons vu dans la Sous-Section 6.6.2 que tout automate acyclique standard et homogène est un automate de Glushkov étendu. Cette propriété s'étend à tous les automates finis standards homogènes comme suit :

Théorème 6.2 *Soit A un NFA homogène et standard. Les deux propositions suivantes sont équivalentes :*

***(1)** l'automate A est un automate de Glushkov étendu,*

***(2)** toute orbite de A est fortement stable et fortement transversale.*

Démonstration. **(1⇒2) :** Par construction depuis une ERA E de l'automate des positions de E isomorphe à A : si une orbite est présente dans A, alors l'expression E admet une sous-expression F^*. Par construction, les symboles précédant les First(F) sont tous reliés dans l'automate A à tous les symboles de First(F), entrée de l'orbite. De même, tout symbole de Last(F), sortie de l'orbite de A, est relié dans l'automate à tout symbole suivant les Last(F), et par l'étoile aux entrées de l'orbite, les symboles de

First(F). Par induction sur les étoiles imbriquées, toute orbite de A est fortement stable et fortement transversale.

(1⇐2) : Si A est acyclique, selon la Proposition 6.17, l'automate A est un automate de Glushkov étendu. Sinon, nous procédons par induction sur les orbites maximales de A. Le cas de base pour l'induction est le suivant : Soit O une orbite maximale de A. Puisque toute orbite maximale O de A est fortement stable et fortement transversale, on élimine les arcs de $\mathrm{Out}(O) \times \mathrm{In}(O)$. Une fois les arcs de $\mathrm{Out}(O) \times \mathrm{In}(O)$ éliminés, supposons alors que l'automate obtenu soit acyclique. Considérons alors l'automate $A' = (\Sigma, Q', I, F', \delta')$ défini par :

(-) $Q' = O \cup \{O^-\}$,
(-) $I' = \{O^-\}$,
(-) $F' = \mathrm{Out}(O)$,
(-) $\delta' = \{(p,a,q) \in (O \times \Sigma \times O) \cap \delta\}$
$\cup \{(p,a,q) \in \{O^-\} \times \Sigma \times O \mid q \in \mathrm{In}(O) \ \wedge \ \exists p' \in O^- \mid (p',a,q) \in \delta\}$.

La construction de l'automate A' est illustrée Figure 6.16, Figure 6.17 et Figure 6.18. Selon la Proposition 6.17, il existe une ERA E_O dénotant le langage reconnu par A', de largeur $\#Q' - 1$ et telle que l'automate des positions de E_O soit isomorphe à A'. Puisque O est fortement stable et fortement transversale, on peut alors remplacer les sommets de O par le sommet $s = \overline{E_O^*}$ et modifier les transitions appartenant à $O^- \times \mathrm{In}(O)$ et à $\mathrm{Out}(O) \times O^+$ respectivement par $O^- \times \{s\}$ et par $\{s\} \times O^+$. On transforme ainsi une orbite de taille k en une expression étoilée de taille k. Et l'orbite O est éliminée.

Par induction sur les orbites, on obtient alors une ERA E de largeur n avec n le nombre d'états de A, dénotant $L(A)$, telle que l'automate des positions de E soit isomorphe à A. □

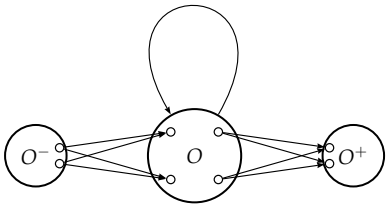


FIGURE 6.16 – *Une Orbite O de A.*

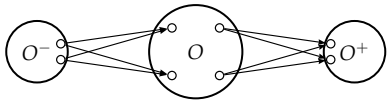


FIGURE 6.17 – *Le Graphe Acyclique de O.*

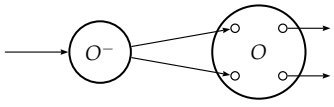


FIGURE 6.18 – *L'Automate A′.*

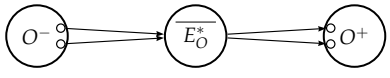


FIGURE 6.19 – *Réduction du Graphe.*

Exemple 6.12 *Considérons l'automate A représenté Figure 6.20. Cet automate possède une orbite O_1 maximale fortement stable et fortement transversale contenant les états*

$\{1,2,3,4,5,6\}$. *En éliminant les arcs de* $\mathrm{Out}(O_1) \times \mathrm{In}(O_1)$, *l'automate produit contient une orbite* $O_2 = \{2,3,4,5\}$ *également fortement stable et fortement transversale (voir Figure 6.21). En éliminant les transitions de* $\mathrm{Out}(O_2)$ *vers* $\mathrm{In}(O_2)$, *l'automate produit est acyclique (voir Figure 6.22). On peut alors réduire l'automate correspondant à l'orbite* O_2 *(voir Figure 6.23), en une expression* $E_{O_2} = (b \cdot (\overbrace{\widetilde{c}\ \widetilde{d}}) \cdot e)$ *et intégrer cette expression dans A. L'automate A devient alors l'automate A′ représenté dans la Figure 6.24, contenant une orbite* $O'_1 = \{1, 6, \overline{E^*_{O_2}}\}$. *En éliminant les transitions de* $\mathrm{Out}(O'_1) \times \mathrm{In}(O'_1)$, *l'automate produit est acyclique (voir Figure 6.25). On peut alors réduire l'automate correspondant à l'orbite* O'_1, *en une expression* $E_{O'_1} = a \cdot \overline{E^*_{O_2}} \cdot f$ *et intégrer cette expression dans A′. L'automate A′ devient alors l'automate acyclique A″ représenté dans la Figure 6.26. On obtient alors l'ERA* $E = \overline{(a \cdot (\overline{E^*_{O_2}}) \cdot f)^*}$, *avec* $E_{O_2} = (b \cdot (\overbrace{\widetilde{c}\ \widetilde{d}}) \cdot e)$ *reconnaissant le langage* $L(A)$.

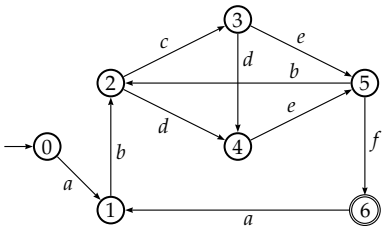


FIGURE 6.20 – *L'Automate A.*

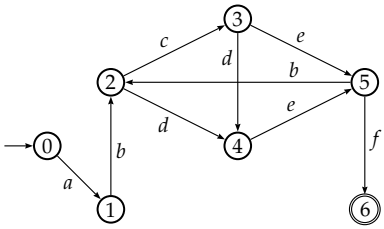


FIGURE 6.21 – *Élimination de la Transition* $(6, a, 1)$.

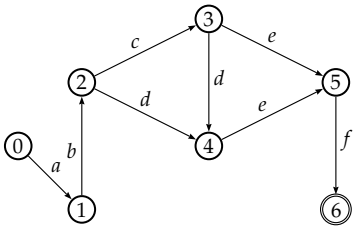


FIGURE 6.22 – *Élimination de la Transition* $(5, b, 2)$.

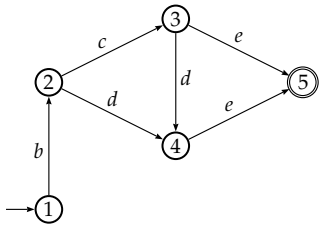


FIGURE 6.23 – *L'Automate Correspondant à* O_2.

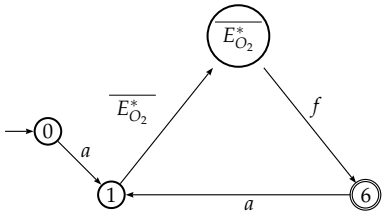


FIGURE 6.24 – *L'Automate A'.*

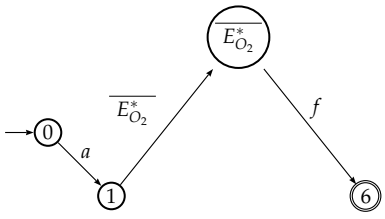


FIGURE 6.25 – *Élimination de la Transition* $(6, a, 1)$.

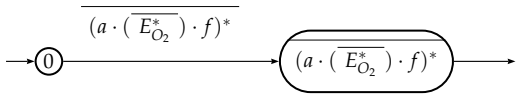


FIGURE 6.26 – *L'Automate A''.*

6.7 Conclusion

La définition des notions de dérivées, dérivées partielles et de c-continuations assurent un calcul efficace d'un automate à partir d'une ERA quelconque. La caractérisation des automates de Glushkov étendus nous permet d'augmenter le nombre d'automates réductibles rapidement vers une expression concise. Nous avons ainsi montré comment étendre des méthodes de conversion de et vers les expressions rationnelles simples aux ERA, et que les ERA sont des structures de calculs efficaces permettant de faire le lien entre les automates et les expressions rationnelles simples.

Nous montrerons dans le chapitre suivant que les expressions à contraintes induisent un pouvoir de réduction exponentiellement plus grand que les expressions rationnelles simples.

Chapitre 7

Pouvoir de Factorisation

C'est un fait de pure expérience qu'il n'y a pas
d'espace sans temps ni de temps sans espace.
Daisetz Téitaro Suzuki

La jeunesse veut l'espace ; la vieillesse, le temps.
Jean Nohain

Sommaire

Dans ce chapitre, nous donnons des exemples illustrant le pouvoir de factorisation des ERABT et des ERA. Dans la Section 7.1, nous présentons deux familles de langages dénotées l'une par une famille de multi-barres, l'autre par une famille de multi-tildes. Nous montrons que chacun de ces langages est dénoté par une ERABT de largeur k et par une expression simple de taille exponentielle en fonction de k. Dans la Section 7.2, nous utilisons une famille de langages déjà étudiée par Ehrenfeucht et Zeiger ([22] et [23]), les langages $(H_k)_{k\geq 3}$. Pour chaque langage H_k de cette famille, il existe un NFA à k états reconnaissant H_k et toute expression rationnelle simple E_k dénotant H_k est telle que $|E_k| = k^{\Omega(\log\log k)}$. Nous montrons que H_k est reconnu par une ERA de largeur $\frac{k(k+1)}{2}$.

7.1 Pouvoir de Réduction des ERABT

Dans cette section, nous étudions le pouvoir de factorisation des ERABT sur des exemples paramétrés. Les multi-barres et les multi-tildes conduisent à une réduction de la largeur alphabétique généralement plus importante que par une factorisation directe d'une expression rationnelle simple équivalente. Nous illustrons cela par l'Exemple 7.1 pour les multi-barres et l'Exemple 7.2 pour les multi-tildes.

Exemple 7.1 *Soit* $E_{B_3} = \overline{}^{\left((1,2),(2,3)\right)}\left(a+\varepsilon, b+\varepsilon, c+\varepsilon\right)$ *une ERABT dont la représentation graphique est* $\overline{(a+\varepsilon)(b+\varepsilon)(c+\varepsilon)}$ *. Sa largeur alphabétique est égale à* 3*. Le langage dénoté par* E_{B_3} *est reconnu par l'automate de la Figure 7.2. Remarquons que toute expression rationnelle simple est plus large que* E_{B_3}*. Par exemple, la largeur de l'expression* $E'_{B_3} = ac + (a+\varepsilon)b(c+\varepsilon)$ *est égale à* 5*.*

Exemple 7.2 *Soit* $E_{T_3} = \sim_{(1,2),(2,3)}(a,b,c)$ *une ERABT de largeur* 3 *représentée graphiquement par* $E_{T_3} = \widetilde{a\ b\ c}$ *. Le langage* $L(E_{T_3})$ *est reconnu par l'automate de la Figure 7.1. Toute expression rationnelle simple équivalente est plus large que* E_{T_3}*, comme par exemple l'expression* $E'_{T_3} = a(bc+\varepsilon) + c$ *de largeur* 4*.*

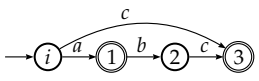


FIGURE 7.1 – *Un Automate Reconnaissant* $L(E_{T_3})$*.*

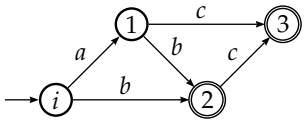


FIGURE 7.2 – *Un Automate Reconnaissant* $L(E_{B_3})$*.*

Nous généralisons l'Exemple 7.1 et l'Exemple 7.2 en considérant les familles de langages rationnels $\{L(E_{B_n}) \mid n \geq 3\}$ et $\{L(E_{T_n}) \mid n \geq 3\}$ avec :

$$\forall n \geq 3,\ E_{B_n} = \overline{}^{B}(F_{1,n}),$$
$$E_{T_n} = \sim_{T_n}(G_{1,n}),$$

où pour tout entier k supérieur ou égal à 3, $F_k = a_k + \varepsilon$, $G_k = a_k$, et $B_k = T_k = \{(k', k'+1) \mid k' \in [\![1, k-1]\!]\}$. Soit k un entier supérieur ou égal à 3. Les expressions E_{B_k} et E_{T_k} sont représentées graphiquement par :

$$E_{B_k} = \overline{(a_1+\varepsilon)\overline{(a_2+\varepsilon)}\overline{(\cdots)}\overline{(a_{k-1}+\varepsilon)\,(a_k+\varepsilon)}}\ ,\ E_{T_k} = \widetilde{a_1\ a_2 \ldots a_{k-1}\ a_k}\ .$$

Pour illustrer le pouvoir de factorisation des opérateurs de multi-barres (resp. des opérateurs de multi-tildes), nous comparons la largeur de E_{B_n} (resp. de E_{T_n}) égale à n avec la largeur minimale d'une expression simple E'_{B_n} (resp. E'_{T_n}) dénotant $L(E_{B_n})$ (resp. $L(E_{T_n})$). Malheureusement, pour un langage rationnel, il n'existe pas d'algorithme polynomial pour calculer une expression rationnelle simple minimale, même dans le cas des langages finis. Nous procédons alors comme suit : nous commençons depuis les expressions $E'_{B_3} = a_1a_3 + (\varepsilon + a_1)a_2(\varepsilon + a_3)$ et $E'_{T_3} = a_1(a_2a_3 + \varepsilon) + a_3$ qui

sont minimales, et nous appliquons une transformation particulière pour construire récursivement E'_{B_n} depuis $E'_{B_{n-1}}$ et E'_{T_n} depuis $E'_{T_{n-1}}$. Bien que les minimalités de E'_{B_n} et E'_{T_n} ne soient prouvées, ces expressions semblent être de bons candidats pour la minimalité. Nous montrons alors que les expressions simples E'_{B_n} et E'_{T_n} admettent des largeurs exponentielles en fonction de n. Remarquons que chacune des familles des langages est définie sur un alphabet de taille non constante. Pour prouver la construction que nous mettons en place, nous utilisons les propriétés suivantes :

Lemme 7.1 *Soit w un mot de $L(F_{1\cdots n})$. Les quatre propositions suivantes sont équivalentes :*

(1) $w \in L(E_{B_n})$,
(2) $\forall k \in [\![1, k-1]\!], w_k \cdot w_{k+1} \neq \varepsilon$,
(3) $w \cdot a_{n+1} \in L(E_{B_{n+1}})$,
(4) *Si* $w_n = a_n$ *alors* $w \in L(E_{B_{n+1}})$.

Démonstration. Procédons par suite d'implications :

(1 ⇒ 2) Si w est un mot de $L(E_{B_n})$, selon la Définition 3.2, il admet une décomposition vérifiant toutes les barres de B_n. Ainsi, puisque pour tout k de $[\![1, n-1]\!]$, $(k, k+1) \in B_n$, on a $w_k \cdot w_{k+1} \neq \varepsilon$.

(2 ⇒ 3) Posons $w' = w'_1 \cdots w'_{n+1}$ tel que pour tout $k \in [\![1, n]\!]$, $w'_k = w_k$ et $w'_{n+1} = a_{n+1}$. Supposons $\forall k \in [\![1, k-1]\!], w_k \cdot w_{k+1} \neq \varepsilon$. Ainsi, w' vérifie toutes les barres de B_n. De plus, puisque $w'_{n+1} = a_{n+1}$, la barre $(n, n+1)$ de B_{n+1} est vérifiée par w'. Par conséquent, selon la Définition 3.2, le mot $w' = w \cdot a_{n+1}$ est dans $L(E_{B_{n+1}})$.

(3 ⇒ 4) Posons $w' = w'_1 \cdots w'_{n+1}$ tel que pour tout $k \in [\![1, n]\!]$, $w'_k = w_k$ et $w'_{n+1} = a_{n+1}$. Supposons $w' \in L(E_{B_{n+1}})$. Selon la Définition 3.2, toutes les barres de B_{n+1} sont vérifiées par w', c'est à dire que pour tout couple $(k, k+1)$ de $[\![1, n+1]\!]^2_{\leq}$, $w'_k \cdot w'_{k+1} \neq \varepsilon$. Le mot w admet une décomposition $w_1 \cdots w_{n+1}$ avec pour tout $k \in [\![1, n]\!]$, $w_k = w'_k$ et $w_{n+1} = \varepsilon$. Si $w_n = a_n$, alors le mot w vérifie la barre $(n, n+1)$, puisque $w_n \cdot w_{n+1} = a_n$. De plus, il vérifie également toutes les barres de B_n (sinon, contradiction avec w' dans $L(E_{B_{n+1}})$). Par conséquent, $w \in L(E_{B_{n+1}})$.

(4 ⇒ 1) Supposons $w \in L(E_{B_{n+1}})$. Selon la Définition 3.2, toutes les barres de B_{n+1} sont vérifiées par le mot w, c'est à dire que pour tout couple $(k, k+1)$ de $[\![1, n+1]\!]^2_{\leq}$, $w_k \cdot w_{k+1} \neq \varepsilon$. Puisque $B_n \subset B_{n+1}$ et que toutes les barres de B_{n+1} sont vérifiées par w, w vérifie également toutes les barres de B_n. Selon la Définition 3.2, $w \in L(E_{B_n})$. □

Lemme 7.2 *Soit $w = w_1 \cdots w_n$ un mot tel que $\forall k \in [\![1, n]\!]$, $w_k \in L(G_k) \cup \{\varepsilon\}$. Les trois propositions suivantes sont équivalentes :*

(1) $w \in L(E_{T_n})$,
(2) $w \in L(E_{T_{n+2}})$,
(3) $w \cdot a_{n+1} \in L(E_{T_{n+1}})$.

Démonstration. Procédons par une suite d'implications :

(1⇒2) Si w est dans $L(E_{T_n})$, selon le Lemme 3.11, il existe une sous-liste libre T' de T_n telle que $w \in L(\sim_{T'}(G_{1,n}))$. Puisque la liste de couples $T_{n+2} = T_n \cup \{(n, n+1), (n+1, n+2)\}$, la liste $T'' = T' \cup \{(n+1, n+2)\}$ est une sous-liste libre de T_{n+2}. Ainsi, puisque $w \in L(\sim_{T''}(G_{1,n+2}))$ et que $L(\sim_{T''}(G_{1,n+2})) \subset L(E_{T_{n+2}})$, $w \in L(E_{T_{n+2}})$.

(2⇒3) Si w est dans $L(E_{T_{n+2}})$, selon le Lemme 3.11, il existe une sous-liste libre T' de T_{n+2} telle que $w \in L(\sim_{T'}(G_{1,n+2}))$. Le mot w admet donc une décomposition $w_1 \cdots w_{n+2}$ sur $(G_1 + \varepsilon) \cdots (G_{n+2} + \varepsilon)$ telle que le mot $w_{n+1} \cdot w_{n+2}$ soit égal à ε par définition de w. Ainsi, la tilde $(n+1, n+2)$ appartient à T' (sinon contradiction avec w dans $L(E_{T_{n+2}})$). Puisque T' est une sous-liste libre, $T'' = T' \setminus \{(n+1, n+2)\}$ est une sous-liste libre incluse dans T_{n+1}. Par conséquent, puisque $a_{n+1} \in L(G_{n+1})$, le mot $w \cdot a_{n+1}$ est dans $L(\sim_{T''}(G_{1,n+1}))$, langage inclus dans $L(E_{T_{n+1}})$ selon le Lemme 3.11.

(3⇒1) Supposons $w \cdot a_{n+1} \in L(E_{T_{n+1}})$. Selon le Lemme 3.11, il existe une sous-liste libre T' de T_{n+1} telle que $w \in L(\sim_{T'}(G_{1,n+1}))$. Puisque la lettre a_{n+1} est présente à la fin du mot $w \cdot a_{n+1}$, la présence de la tilde $(n, n+1)$ de T_{n+1} n'est pas nécessaire et par conséquent, on peut supposer que le couple $(n, n+1)$ n'est pas dans T'. Finalement, $T' \subset T_n$ et ainsi puisque $w \in L(\sim_{T'}(G_{1,n}))$, selon le Lemme 3.11, $w \in L(E_{T_n})$. □

Considérons un entier n supérieur à 3. Soit $w_1 \cdots w_n$ un mot de $L(E_{B_n})$ (resp. $L(E_{T_n})$). Puisque pour tout entier k de $[\![1, n]\!]$, on a soit $w_k = a_k$ soit $w_k = \varepsilon$, $L(E_{B_n})$ (resp. $L(E_{T_n})$) peut être représenté par un arbre D_{B_n} (resp. D_{T_n}) ressemblant à un arbre binaire de décision. Par exemple, l'arbre D_{T_3} de $L(E_{T_3}) = \{a_1, a_3, a_1a_2a_3\}$ est représenté par la Figure 7.3.

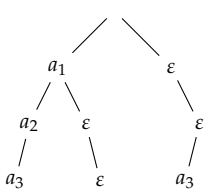


FIGURE 7.3 – *L'Arbre D_{T_3}.*

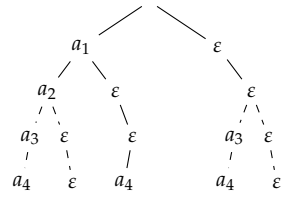


FIGURE 7.4 – *L'Arbre D_{T_4}.*

La Figure 7.5 illustre le cas de E_{B_3}. Il existe deux sous-arbres identiques (en pointillés), et ainsi il est possible d'effectuer une factorisation de l'arbre D_{B_3}. Après avoir éliminé les chemins inutiles, nous obtenons l'arbre D'_{B_3} de E'_{B_3} (Figure 7.6). L'expression E'_{B_3} est calculée par un parcours particulier de cet arbre, appelé *construction préfixe*.

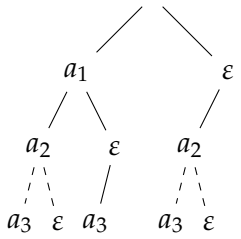


FIGURE 7.5 – *L'Arbre D_{B_3}.*

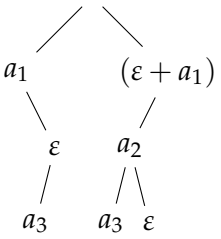


FIGURE 7.6 – *L'Arbre D'_{B_3}.*

Définition 7.1 *Soit D un arbre binaire, soit D_g (resp. D_d) son sous-arbre gauche (resp. droit). La* construction préfixe *de l'expression de l'arbre se définit récursivement par :*

$E' = r \cdot (E'_g + E'_d)$ *si D_g et D_d ne sont pas des arbres vides*
$E' = r \cdot (E'_g)$ *si D_g n'est pas un arbre vide et D_d est un arbre vide*
$E' = r \cdot (E'_d)$ *si D_d n'est pas un arbre vide et D_g est un arbre vide*
$E' = r$ *sinon*

avec r la valeur de la racine,
E'_g (resp. E'_d) le résultat de la construction préfixe de D_g (resp. D_d).

Par construction préfixe de l'expression de D'_{B_3}, on obtient l'expression $E'_{B_3} = a_1a_3 + (\varepsilon + a_1)a_2(\varepsilon + a_3)$. On peut aisément vérifier que les langages $L(E'_{B_3})$ et $L(E'_{B_3})$ sont égaux.

L'expression simple $E'_{T_3} = a_1(a_2a_3 + \varepsilon) + a_3$ obtenue par construction préfixe à partir de D_{T_3} est équivalente à E_{T_3} et a une largeur minimale. L'arbre D_{T_3} peut être aisément transformé en l'arbre D_{T_4} (voir Figure 7.4) et son expression associée est $E''_{T_4} = a_1(a_2(a_3a_4 + \varepsilon) + a_4) + a_3a_4 + \varepsilon$ qui est équivalente à E_{T_4}. La structure d'arbre induit uniquement une factorisation gauche et l'expression E''_{T_4} n'est pas minimale. En effet, les deux sous-arbres en pointillés de la Figure 7.4 sont identiques, mise à part l'étiquette de la racine. Cela implique qu'une meilleure factorisation existe qui conduit à l'arbre D'_{T_4} de la Figure 7.7 et à l'expression obtenue par construction préfixe $E'_{T_4} = a_1a_4 + (a_1a_2 + \varepsilon)(a_3a_4 + \varepsilon)$.

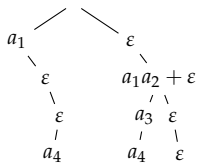


FIGURE 7.7 – *L'Arbre D'_{T_4}.*

Nous définissons alors une génération récursive de E'_{B_n} et E'_{T_n}. Nous séparons les arbres les plus profonds de taille 3 des arbres D'_{B_n} et D'_{T_n} en trois catégories (voir Table 7.1 et Table 7.2).

Un sous-arbre de type 3 contient un sous-arbre de type 1 et un de type 2. Cependant, dans la Table 7.1 et dans la Table 7.2, il existe une meilleure

factorisation droite que celle obtenue par les transformations de type 1 et type 2.

En utilisant cette construction, nous calculons alors étape par étape les expressions rationnelles simples E'_{B_n} et E'_{T_n}. En exemple, voici les expressions E'_{B_n} et E'_{T_n} pour k dans $[\![3,6]\!]$:

$$\begin{aligned}
E'_{B_3} &= a_1a_3 + (\varepsilon + a_1)a_2(\varepsilon + a_3),\\
E'_{B_4} &= a_1a_3(\varepsilon + a_4)\\
&\quad +(\varepsilon + a_1)a_2(a_4 + a_3(\varepsilon + a_4)),\\
E'_{B_5} &= a_1a_3(a_5 + a_4(\varepsilon + a_5))\\
&\quad +(\varepsilon + a_1)a_2(a_3a_5 + (a_3 + \varepsilon)a_4(\varepsilon + a_5)),\\
E'_{B_6} &= a_1a_3(a_4a_6 + (\varepsilon + a_4)a_5(\varepsilon + a_6))\\
&\quad +(\varepsilon + a_1)a_2(a_3a_5(\varepsilon + a_6) + (a_3 + \varepsilon)a_4(a_6 + a_5(\varepsilon + a_6))),\\
E'_{T_3} &= a_1(a_2a_3 + \varepsilon) + a_3,\\
E'_{T_4} &= a_1a_4\\
&\quad +(a_1a_2 + \varepsilon)(a_3a_4 + \varepsilon),\\
E'_{T_5} &= a_1(a_4a_5 + \varepsilon)\\
&\quad +(a_1a_2 + \varepsilon)(a_3(a_4a_5 + \varepsilon) + a_5),\\
E'_{T_6} &= a_1(a_4(a_5a_6 + \varepsilon) + a_6)\\
&\quad +(a_1a_2 + \varepsilon)(a_3a_6 + (a_3a_4 + \varepsilon)(a_5a_6 + \varepsilon)).
\end{aligned}$$

Montrons alors que les expressions E'_{B_n} (resp. E'_{T_n}) et E_{B_n} (resp. E_{T_n}) sont équivalentes.

Lemme 7.3 *$L(E'_{B_n}) = L(E_{B_n})$ et $L(E'_{T_n}) = L(E_{T_n})$.*

Démonstration. Ce lemme est prouvé par induction sur n.

$L(E'_{B_n}) = L(E_{B_n})$: Au rang $n = 3$, on peut facilement vérifier que les langages $L(E'_{B_3})$ et $L(E_{B_3})$ sont égaux. Supposons la propriété vérifiée pour $n \geq 3$. Soit w un mot de $L(E'_{B_{n+1}})$. Par construction de $E'_{B_{n+1}}$, pour tout entier k de $[\![1,n]\!]$, le mot $w_k \cdot w_{k+1}$ est différent du mot vide ε. Selon le Lemme 7.1, w est un mot de $L(E_{B_{n+1}})$. Réciproquement, soit w un mot de $L(E_{B_{n+1}})$. Deux cas sont à traiter. Si $w = w' \cdot w_n a_{n+1}$, alors $w' \cdot w_n$ est dans $L(E_{B_n})$ (selon le Lemme 7.1). Par hypothèse d'induction, $w' \cdot w_n$ est dans $L(E'_{B_n})$ et par construction w appartient à $L(E'_{B_{n+1}})$. Si $w = w' \cdot a_n$, alors w est dans $L(E_{B_n})$ (selon le Lemme 7.1). Par hypothèse d'induction, w est dans $L(E'_{B_n})$. Et ainsi, par la construction de l'arbre, w appartient à $L(E'_{B_{n+1}})$.

$L(E'_{T_n}) = L(E_{T_n})$: Pour les rangs $n = 3$ et $n = 4$, on peut vérifier que les langages $L(E'_{T_n})$ et $L(E_{T_n})$ sont égaux. Supposons cette propriété vraie au rang $n \geq 4$. Soit w un mot de $L(E_{T_{n+1}})$. Selon le Lemme 7.2, le mot w est dans $L(E_{T_{n-1}})$. Selon l'hypothèse d'induction, w est dans $L(E'_{T_{n-1}})$. Ainsi, par construction, w appartient à $L(E'_{T_{n+1}})$. Réciproquement, supposons que w soit un mot de $L(E'_{T_{n+1}})$. Deux cas sont à traiter. Si $w_{n+1} = a_{n+1}$, par construction, $w_1 \cdots w_n$ est dans $L(E'_{T_n})$. Selon l'hy-

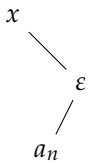
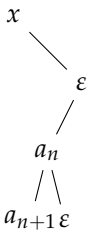
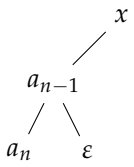
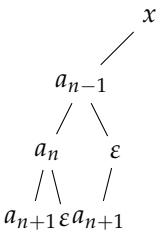
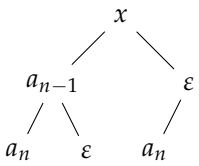
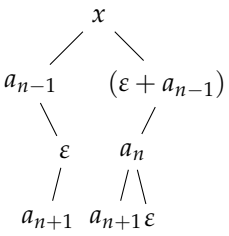
Catégorie	Sous-arbre	Transformation
Type 1		
Type 2		
Type 3		

TABLE 7.1 – *Catégories et Transformations Associées pour* D'_{B_n}.

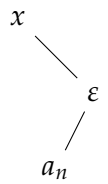
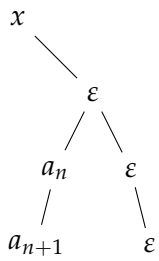
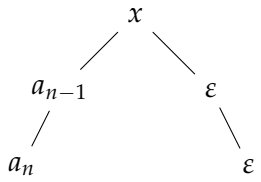
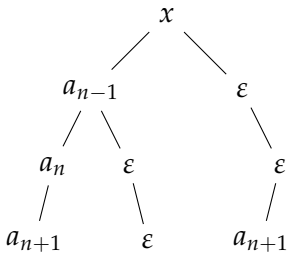
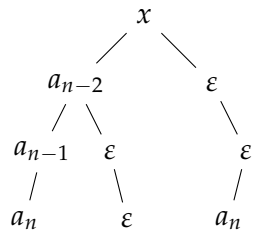
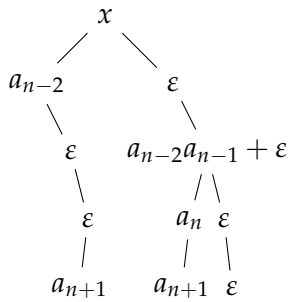
Catégorie	Sous-arbre	Transformation
Type 1		
Type 2		
Type 3		

TABLE 7.2 – *Catégories et Transformations Associées pour* D'_{T_n}.

pothèse d'induction, $w_1 \cdots w_n$ est dans $L(E_{T_n})$. Et selon le Lemme 7.2, le mot $w = w_1 \cdots w_n \cdot a_{n+1}$ appartient à $L(E_{T_{n+1}})$. Si $w_{n+1} = \varepsilon$, par construction, $w_n = \varepsilon$, et $w = w_1 \cdots w_{n-1}$ est dans $L(E'_{T_{n-1}})$. Selon l'hypothèse d'induction, w est dans $L(E_{T_{n-1}})$, et selon le Lemme 7.2, w appartient à $L(E_{T_{n+1}})$. □

La largeur de E'_{B_n} (resp. E'_{T_n}) est égale au nombre $\mathrm{t}(n)$ de lettres de l'arbre D'_{B_n} (resp. D'_{T_n}). Notons respectivement $\mathrm{k}_1(n)$, $\mathrm{k}_2(n)$ et $\mathrm{k}_3(n)$ le nombre de sous-arbres de type 1, de type 2 et de type 3 de D'_{B_n} (resp. D'_{T_n}).

Lemme 7.4 *Les nombres* $\mathrm{t}(n)$, $\mathrm{k}_1(n)$, $\mathrm{k}_2(n)$ *et* $\mathrm{k}_3(n)$ *se calculent récursivement comme suit :*
Pour D'_{B_n} *:*

$$\mathrm{k}_1(n+1) = \begin{cases} 1 & \text{si } n = 2 \\ \mathrm{k}_3(n) & \text{sinon} \end{cases} \quad \mathrm{k}_2(n+1) = \begin{cases} 1 & \text{si } n = 2 \\ \mathrm{k}_1(n) + \mathrm{k}_3(n) & \text{sinon} \end{cases}$$

$$\mathrm{k}_3(n+1) = \begin{cases} 0 & \text{si } n = 2 \\ \mathrm{k}_2(n) & \text{sinon} \end{cases} \quad \mathrm{t}(n+1) = \begin{cases} 5 & \text{si } n = 2 \\ \mathrm{t}(n) + \mathrm{k}_1(n) + 2\mathrm{k}_2(n) + 2\mathrm{k}_3(n) & \text{sinon} \end{cases}$$

Pour D'_{T_n} *:*

$$\mathrm{k}_1(n+1) = \begin{cases} 0 & \text{si } n = 2 \\ \mathrm{k}_3(n) & \text{sinon} \end{cases} \quad \mathrm{k}_2(n+1) = \begin{cases} 0 & \text{si } n = 2 \\ \mathrm{k}_1(n) + \mathrm{k}_3(n) & \text{sinon} \end{cases}$$

$$\mathrm{k}_3(n+1) = \begin{cases} 1 & \text{si } n = 2 \\ \mathrm{k}_2(n) & \text{sinon} \end{cases} \quad \mathrm{t}(n+1) = \begin{cases} 4 & \text{si } n = 2 \\ \mathrm{t}(n) + \mathrm{k}_1(n) + 2\mathrm{k}_2(n) + 2\mathrm{k}_3(n) & \text{sinon} \end{cases}$$

Démonstration. Au rang $n = 3$, les formules sont correctes selon les arbres D'_{B_3} et D_{T_3}. Au rang $n \geq 3$, pour D'_{B_n} et D_{T_n}, une transformation de type 1 génère un sous-arbre de type 2, une transformation de type 2 génère un sous-arbre de type 3 et une transformation de type 3 génère à la fois un sous-arbre de type 1 et un sous-arbre de type 2. D'où les formules pour $\mathrm{k}_1(n+1)$, $\mathrm{k}_2(n+1)$ et $\mathrm{k}_3(n+1)$. De plus, le nombre de lettres générées est de 1 pour les sous-arbres de type 1, et de 2 pour les sous-arbres de type 2 et de type 3. D'où la formule pour $\mathrm{t}(n+1)$. □

Montrons maintenant que les nombres $\mathrm{k}_1(n)$, $\mathrm{k}_2(n)$ et $\mathrm{k}_3(n)$ sont des suites de Padovan (voir Définition 3.7) :

Lemme 7.5 *Les nombres* $\mathrm{k}_1(n)$, $\mathrm{k}_2(n)$ *et* $\mathrm{k}_3(n)$ *sont des suites de Padovan avec décalage.*

Démonstration. Remarquons que pour tout entier k supérieur ou égal à 5, on a $\mathrm{k}_3(n) = \mathrm{k}_2(n-1)$, et $\mathrm{k}_1(n) = \mathrm{k}_3(n-1) = \mathrm{k}_2(n-2)$. Ainsi, pour tout entier k supérieur ou égal à 5, $\mathrm{k}_2(n) = \mathrm{k}_2(n-2) + \mathrm{k}_2(n-3)$. **(1)** Pour D'_{B_n}, $\mathrm{k}_2(3) = \mathrm{k}_2(4) = \mathrm{k}_2(5) = 1$. Finalement, pour tout entier k supérieur ou égal à 3, $\mathrm{k}_2(n) = \mathrm{P}(n-3)$. **(2)** Pour D'_{T_n}, $\mathrm{k}_2(4) = \mathrm{k}_2(5) = \mathrm{k}_2(6) = 1$. D'où pour tout entier k supérieur ou égal à 4, $\mathrm{k}_2(n) = \mathrm{P}(n-4)$. □

Lemme 7.6 *L'expression* E'_{B_n} *(resp.* E'_{T_n}*) admet une largeur exponentielle en fonction de* n*.*

Démonstration. On a $\mathrm{t}(n+1) = \mathrm{t}(n) + \mathrm{k}_2(n-2) + \mathrm{k}_2(n) + \mathrm{k}_2(n-1)$.
En approximant les résultats, on peut voir que :

$$\text{Pour } D'_{B_n} : \mathrm{t}(n) \approx \sum_{k=3}^{n} \mathrm{k}_2(k) \approx \sum_{k=3}^{n} \mathrm{P}(k-3) \approx \sum_{k=3}^{n} \frac{p^k}{4}$$
$$\text{Pour } D'_{T_n} : \mathrm{t}(n) \approx \sum_{k=4}^{n} \mathrm{k}_2(k) \approx \sum_{k=4}^{n} \mathrm{P}(k-4) \approx \sum_{k=4}^{n} \frac{p^k}{4}$$

ce qui implique la formule $\mathrm{t}(n) \approx \dfrac{p^{n+1}-1}{p-1} \approx p^n \approx e^{0.27n}$. □

7.2 Le Gain de Factorisation des ERA est Exponentiel

Dans cette section, nous étudions le pouvoir de factorisation des ERA sur la famille de langages finis $\mathcal{H} = \{H_k \mid k \geq 1\}$ définie et étudiée par Ehrenfeucht et Zeiger dans [22]. Tout langage H_k de $\mathcal{H}$ est l'ensemble des chemins du graphe $G_k = (U_k, V_k)$, avec :

$$U_k = [\![1,k]\!] \text{ et } V_k = \{(i,f) \in [\![1,k]\!]^2_< \},$$

avec pour tout (i,f) de $\mathbb{N}^2$, $[\![i,f]\!]^2_< = \{(i',f') \in [\![1,k]\!]^2 \mid i' < f'\}$.

Définition 7.2 *Soit k un entier positif et $S = ((i_l, f_l)_{l \in [\![1,\#S]\!]})$ une liste de $\mathcal{S}_k$. La liste S est une* liste revêtant $[\![1,k]\!]$ *si les quatre propriétés suivantes sont vérifiées :*

(1) $i_1 = 1$,
(2) $f_{\#S} = k$,
(3) *pour tout l de $[\![1,\#S]\!]$, $i_l < f_l$,*
(4) *pour tout l de $[\![1,\#S-1]\!]$, $f_l = i_{l+1}$.*

Remarquons que V_k est l'ensemble des listes S de $\mathcal{S}_k$ *revêtant* $[\![1,k]\!]$.

Pour transformer une liste $S = ((i_l, f_l)_{l \in [\![1,\#S]\!]})$ de $\mathcal{S}_k$ en un mot w sur l'alphabet $[\![1,k]\!]^2_{\leq}$, nous définissons l'opération $\odot S = (i_1, f_1) \cdots (i_{\#S}, f_{\#S})$. Le mot $w = \odot S$ est appelé *mot associé à S*.

Exemple 7.3 *Soit $S = ((1,2),(2,4),(4,5),(5,7))$ une liste de $\mathcal{S}_7$. La liste S revêt l'intervalle $[\![1,7]\!]$. Le mot associé à S est $\odot S = (1,2) \cdot (2,4) \cdot (4,5) \cdot (5,7)$.*

Définition 7.3 *Soit k un entier positif. Le langage H_k est défini sur l'alphabet $[\![1,k]\!]^2_{\leq}$ par :*

$$H_k = \{\odot S \mid S \in \mathcal{S}_k \ \wedge \ S \text{ revêt } [\![1,k]\!]\}.$$

La famille $\mathcal{H}$ est définie sur un alphabet de taille croissante, et tout langage H_k de $\mathcal{H}$ vérifie le théorème suivant :

Théorème 7.1 *Soit H_k un langage de $\mathcal{H}$. Les trois propriétés suivantes sont vérifiées :*

(1) *H_k est reconnu par un NFA à k états,*
(2) *pour toute expression rationnelle simple E_k :*

$$L(E_k) = H_k \Rightarrow |E_k| = k^{\Omega(\mathrm{loglog}k)},$$

(3) *il existe une ERA E_k dénotant H_k de largeur $\frac{k(k+1)}{2}$.*

Nous montrons alors que tout langage H_k de la famille $\mathcal{H}$ vérifie les propriétés énoncées.

Proposition 7.1 *Soit H_k un langage de $\mathcal{H}$. Considérons le NFA $A_k = (\Sigma, Q, I, F, \delta)$ défini par :*
(-) $\Sigma = [\![1,k]\!]^2_<$,
(-) $Q = [\![1,k]\!]$,
(-) $I = \{1\}$,
(-) $F = \{k\}$,
(-) $\delta = \{(p,a,q) \in Q \times \Sigma \times Q \mid a = (p,q)\}$.
Alors $L(A) = H_k$.

Démonstration. Par construction, tout mot w est dans H_k si et seulement si w est l'étiquette d'un chemin réussi de A. □

Exemple 7.4 *Considérons les langages H_4 et H_5. Le langage H_4 est reconnu par le NFA A_4 à 4 états représenté Figure 7.8. Le langage H_5 est reconnu par le NFA à 5 états représenté Figure 7.9.*

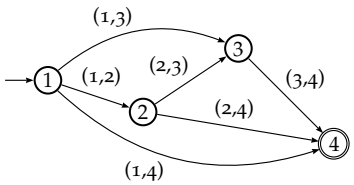


FIGURE 7.8 – *Le NFA A_4 Reconnaissant H_4.*

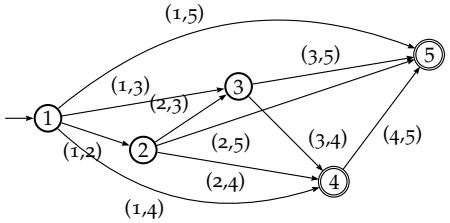


FIGURE 7.9 – *Le NFA A_5 Reconnaissant H_5.*

Proposition 7.2 (Ehrenfeucht et Zeiger [22]) *Soit H_k un langage de $\mathcal{H}$, et E'_k une expression rationnelle simple dénotant H_k. Alors :*

$$|E'_k| = k^{\Omega(\log\log k)}.$$

Pour démontrer l'existence d'une ERA de largeur $\frac{k(k+1)}{2}$ dénotant le langage H_k, nous montrons qu'il existe un automate homogène et standard A'_k reconnaissant H_k de taille $\frac{k(k+1)}{2} + 1$. Nous en déduisons alors selon la Proposition 6.16 l'existence d'une ERA E_k de largeur $\frac{k(k+1)}{2}$.

Proposition 7.3 *Soit H_k un langage de $\mathcal{H}$. Soit l'automate $A'_k = (\Sigma', Q', I', F', \delta')$ défini par :*
(-) $\Sigma' = [\![1,k]\!]^2_<$,
(-) $Q' = \{1_{(0,1)}\} \cup \{f_{(i,f)} \mid (i,f) \in [\![1,k]\!]^2_<\}$,
(-) $I' = \{1_{(0,1)}\}$,
(-) $F' = \{f_{(i,f)} \in Q \mid f = k\}$,
(-) $\delta' = \{(p,(i,f),q') \in Q' \times \Sigma' \times Q' \mid p = i_{(i',i)} \ \wedge \ q = f_{(i,f)}\}$.
L'automate A'_k est standard, homogène et reconnait H_k.

Démonstration. Considérons l'automate A_k de la Proposition 7.1 et l'automate B_k obtenu par homogénéisation de A_k selon l'Algorithme 1.7. Pour tout état f non homogène de A_k, l'algorithme éclate f selon les transitions entrantes. Ainsi, s'il existe i_1 et i_2 deux états de A_k tels que les transitions

$(i_1, (i_1, f), f)$ et $(i_2, (i_2, f), f)$ soient des transitions de A_k, l'algorithme sépare f en deux états $f_{(i_1,f)}$ et $f_{(i_2,f)}$, puis crée les transitions adéquates. Remarquons alors que l'automate A'_k est isomorphe à B_k. Finalement, selon le Lemme 1.16, $L(A'_k) = L(A_k)$. □

Corollaire 7.1 *Soit H_k un langage de $\mathcal{H}$. Alors il existe une ERA E_k dénotant H_k telle que :*

$$|E_k| = \tfrac{k(k+1)}{2}.$$

Démonstration. Considérons l'automate A'_k de la Proposition 7.3. Son ensemble d'états est :

$$Q' = \{1_{(0,1)}\} \cup \{f_{(i,f)} \mid (i, f) \in [\![1, k]\!]^2_< \}.$$

Alors :

$$\#Q' = 1 + \tfrac{k(k-1)}{2}.$$

Selon la Proposition 6.16, il existe une expression E_k dénotant $L(A'_k)$ de largeur $1 + \frac{k(k-1)}{2} - 1 = \frac{k(k-1)}{2}$. □

7.3 Conclusion

Les expressions à contraintes induisent un pouvoir de factorisation exponentiellement plus grand que les expressions rationnelles simples. Le fait que ces expressions soient une représentation intermédiaire entre les expressions rationnelles simples et les automates finis induit ce pouvoir de réduction.

Nous montrerons dans le chapitre suivant que de ce fait découlent également de nombreuses perspectives, puisque certains algorithmes devant s'appliquer sur chacune de ces deux structures particulières pourront être étendus à travers les expressions à contraintes.

Chapitre 8

Conclusion et Perspectives

Une conclusion, c'est quand vous en avez assez de penser.
Herbert Albert Fisher

Si on commence avec des certitudes, on finit avec des doutes.
Si on commence avec des doutes, on finit avec des certitudes.
Francis Bacon

Sommaire

Tout au long de cette thèse, nous avons montré les avantages à disposer du modèle des multi-tildes-barres, se situant entre les expressions rationnelles simples et les automates. Nous avons montré comment obtenir une forme concise pour les ERABT en utilisant l'utilité des couples définissant la Forme Normalisée. Nous avons étendu les notions de multi-barres et de multi-tildes aux multi-tildes-barres, permettant de construire une forme équivalente décrivant plus simplement les langages rationnels, la Forme Totale. Nous avons également montré que les multi-tildes-barres possèdent un pouvoir de factorisation exponentiellement plus important que celui des expressions rationnelles simples, tout en proposant une caractérisation des automates des positions des ERA. De par leur structure particulière, les ERA induisent de nombreuses perspectives de recherche, permettant de transposer les propriétés de chacune des structures d'automate et d'expression rationnelle à l'autre.

8.1 Mise à Jour d'Expressions Rationnelles Simples

Les expressions sont des objets mathématiques très utilisés dans le domaine de la vérification, comme les expressions rationnelles dans la validation de document XML. Bouchou *et al.* proposent dans [6] une méthode de mise à jour d'expressions rationnelles simples en fonction de modifications dénotées sur le langage.

Considérons une expression rationnelle simple E et w un mot de $L(E)$. Soit w' un mot obtenu en ajoutant ou en éliminant un symbole à partir de w. Comment trouver une expression rationnelle simple E' telle que $L(E) \cup \{w'\} \in L(E')$? Une méthode constructive est proposée, par l'algorithme *Generate Regular Expression Choices* (GREC), permettant de calculer un ensemble d'expressions rationnelles E' telles que $L(E')$ inclut $L(E) \cup \{w'\}$ et $(|E'| - |E|)$ est dans $\{-1, 1\}$.

De telles modifications se réalisent très facilement sur des multi-tildes-barres. La suppression d'une lettre d'un mot revient à autoriser la traversabilité du symbole, ce qui correspond à la modification d'une barre en tilde dans l'expression. De par l'action simultanée des tildes et des barres, nous pouvons réduire le nombre de mots ajoutés dans le langage, les tildes augmentant le nombre de sous-intervalles traversables et les barres le diminuant. L'ajout d'un symbole peut être traité de la manière suivante : dans le cas où le symbole n'appartient pas à l'alphabet, on crée un nouvel état dans l'automate des positions de E, et on crée les deux transitions permettant d'inclure ce mot dans le langage, en fonction du mot w'. Si le symbole est déjà un état de l'automate, on doit créer au maximum deux transitions, en fonction de w', pour réaliser la modification du langage souhaitée. Les transitions à ajouter au nouvel automate pour le transformer en un automate de Glushkov étendu sont alors beaucoup plus simples à mettre en oeuvre, puisqu'il suffit d'ajouter certaines transitions pour stabiliser et transversaliser les orbites maximales. L'expression E' obtenue est alors syntaxiquement proche de l'expression de départ E, et une étude approfondie peut permettre de quantifier le degré de modification du langage obtenu.

8.2 Propriétés des Constructions d'Automates

Les familles d'opérateurs de multi-barres, multi-tildes et multi-tildes-barres sont des opérateurs rationnels et compatibles avec la linéarisation. Nous avons également montré comment transformer toute expression E de largeur n utilisant ces opérateurs en un automate A reconnaissant $L(E)$ par les méthodes suivantes : **(1)** une construction directe depuis une ERA sans étoile calculant un automate à $(n+1)$ états ; **(2)** une extension de la construction de l'automate des positions à $(n+1)$ états pour une ERA

quelconque et sa réduction par la congruence des follows de Illie et Yu ; **(3)** une extension des notions de dérivées de Brzozowski et de dérivées partielles d'Antimirov calculant respectivement un DFA A de taille $O(2^{n+1})$ et un NFA A' de taille inférieure à $(n+2)$ tel que l'automate déterministe des parties de A' soit isomorphe à A ; **(4)** une extension de la notion de c-dérivation de Champarnaud et Ziadi construisant un automate isomorphe à l'automate des positions de E. Les extensions (2), (3) et (4) induisent des aspects théoriques intéressants.

(3) et (4) Dans [18], Champarnaud et Ziadi ont montré que l'automate des dérivées partielles est un quotient de l'automate des positions. Ce résultat est-il conservé pour les automates des positions et des équations des ERA ?

(2) et (4) Dans [39], Nicaud a montré qu'en moyenne l'automate des positions d'une expression rationnelle simple E possède $O(n)$ transitions. Dans [7], Broda *et al.* ont montré qu'en moyenne l'automate des équations d'une expression rationnelle simple E possède $(\frac{n}{2})$ états. Qu'en est-il des ERA ? Ces résultats sont-ils conservés ?

Quelles sont les complexités des constructions sur les ERA ? Comment peut on faire l'économie de la forme totale, ou de la (i,f)-traversabilité ? Et que peut-on dire sur le nombre moyen de couples (tildes et barres) des expressions en multi-tildes-barres ? Existe-t-il un moyen de minimiser le nombre de couples pour dénoter un langage, en étendant par exemple la notion d'utilité aux multi-tildes-barres mais aussi aux expressions non-linéaires ?

8.3 Les Multi-Tildes à Contraintes

Le langage d'une multi-tildes-barres $E = \overline{\sim}_{T;B}(E_{1,n})$ est l'union des langages générés par toutes les sous-listes libres de T restreintes par la liste de barres B. Supposons qu'il existe trois tildes consécutives $t_1 = (i_1, f_1)$, $t_2 = (f_1+1, f_2)$ et $t_3 = (f_2+1, f_3)$ dans T telle que la barre (i_1, f_3) soit dans B. Soit $T' = (t'_k)_{k\in[\![1,\#T']\!]}$ une sous-liste libre de T. On peut associer à T' une liste de variables booléennes $\Pi_{T'} = (\pi_{t_k})_{t_k\in T}$ telle que toute variable π_{t_k} prend la valeur VRAI si t_k est dans T', FAUX sinon. L'action de la barre (i_1, f_3) est d'interdire (entre autre) l'utilisation simultanée de t_1, t_2 et t_3 ; pour que la liste de tildes T' soit prise en compte, il faut alors qu'en considérant les variables π_{t_1}, π_{t_2}, et π_{t_3} de $\Pi_{T'}$, la condition booléenne $\neg(\pi_{t_1} \wedge \pi_{t_2} \wedge \pi_{t_3})$ soit égale à VRAI.

Une barre ne peut restreindre l'action de tildes que si elles sont consécutives. On veut maintenant pouvoir interdire l'application simultanée de plusieurs tildes non-nécessairement consécutives et étudier l'augmentation du pouvoir de factorisation de ces nouveaux opérateurs. Par exemple, considérons le langage rationnel L défini par :

$$L = \{w_a \cdot b \cdot w_c \in \{\varepsilon, a\} \times \{b\} \times \{\varepsilon, c\} \mid \neg(w_a = \varepsilon \wedge w_c = \varepsilon)\}.$$

Le langage L est dénoté par l'expression $E = \widetilde{a}\,bc + ab\widetilde{c}$ de largeur 6 et par l'expression $F = ab\widetilde{c} + bc$ de largeur 5. Nous pouvons définir un nouvel opérateur de multi-tildes à contraintes, paramétré par deux listes : la liste de tildes $T = (t_1, t_2)$ avec $t_1 = (1,1)$ et $t_2 = (3,3)$, et la liste de *contraintes* $B = (\neg(t_1 \wedge t_2))$. Son application sur la liste d'expressions (a, b, c) forme une expression $G = \widetilde{\ }_{T;B}(a, b, c)$ de largeur 3.

Ces langages étant rationnels, quel serait alors le pouvoir de factorisation induit par ces nouveaux opérateurs par rapport aux multi-tildes-barres ? Et comment pourrions nous obtenir un automate reconnaissant ce langage ? Quelle serait sa taille ?

Bibliographie

[1] J.-H. Ahn and Y.-S. Han. Implementation of state elimination using heuristics. In Sebastian Maneth, editor, *CIAA*, volume 5642 of *Lecture Notes in Computer Science*, pages 178–187. Springer, 2009.

[2] V.M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2) :291–319, 1996.

[3] D.N. Arden. Delayed-logic and finite-state machines. In *FOCS*, pages 133–151. American Institute of Electrical Engineers, 1961.

[4] G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48(3) :117–126, 1986.

[5] Norbert Blum. An $O(n\log n)$ implementation of the standard method for minimizing n-state finite automata. *Information Processing Letters*, 57(2) :65–59, 1996.

[6] B. Bouchou, D. Duarte, M. Halfeld Ferrari Alves, D. Laurent, and M.A. Musicante. Conservative extensions of regular languages. In *SCCC*, pages 99–109. IEEE Computer Society, 2004.

[7] S. Broda, A. Machiavelo, N. Moreira, and R. Reis. On the average number of states of partial derivative automata. In *Accepté à DLT 2010*, 2010.

[8] A. Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2) :197–213, 1993.

[9] J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In *Mathematical Theory of Automata*, pages 529–561. Polytechnic Press, Polytechnic Institute of Brooklyn, N.Y., 1962. Volume 12 of MRI Symposia Series.

[10] J. A. Brzozowski and E. J. McCluskey. Signal flow graph techniques for sequential circuit state diagrams. *IEEE Transactions on Electronic Computers*, EC-12(2), 1963.

[11] J.A. Brzozowski. Derivatives of regular expressions. *Journal of the Association for Computing Machinery*, 11(4) :481–494, 1964.

[12] P. Caron and M. Flouret. Glushkov construction for multiplicities. In Sheng Yu and Andrei Paun, editors, *CIAA*, volume 2088 of *Lecture Notes in Computer Science*, pages 67–79. Springer, 2000.

[13] P. Caron and D. Ziadi. Characterization of Glushkov automata. *Theoretical Computer Science*, 233(1-2) :75–90, 2000.

[14] J-M. Champarnaud, E. Laugerotte, F. Ouardi, and D. Ziadi. From regular weighted expressions to finite automata. *Int. J. Found. Comput. Sci.*, 15(5) :687–700, 2004.

[15] J-M. Champarnaud, F. Ouardi, and D. Ziadi. An efficient computation of the equation $\mathbb{K}$-automaton of a regular $\mathbb{K}$-expression. *Fundam. Inform.*, 90(1-2) :1–16, 2009.

[16] J.-M. Champarnaud and D. Ziadi. From c-continuations to new quadratic algorithms for automaton synthesis. *Internat. J. Algebra Comput.*, 11(6) :707–735, 2001.

[17] J.-M. Champarnaud and D. Ziadi. From Mirkin's prebases to Antimirov's word partial derivatives. *Informatica Fundamentae*, 45(3) :195–205, 2001.

[18] J.-M. Champarnaud and D. Ziadi. Canonical derivatives, partial derivatives, and finite automaton constructions. *Theoretical Computer Science*, 239(1) :137–163, 2002.

[19] C.-H. Chang and R. Paige. From regular expressions to DFA's using compressed NFA's. In Alberto Apostolico, Maxime Crochemore, Zvi Galil, and Udi Manber, editors, *CPM*, volume 644 of *Lecture Notes in Computer Science*, pages 90–110. Springer, 1992.

[20] C.-H. Chang and R. Paige. From regular expressions to DFA's using compressed NFA's. *Theor. Comput. Sci.*, 178(1-2) :1–36, 1997.

[21] N. Chomsky. Three models for the description of language. *IRE Trans. on Information Theory*, 2(3) :113–124, 1956.

[22] A. Ehrenfeucht and H.P. Zeiger. Complexity measures for regular expressions. In *STOC*, pages 75–79. ACM, 1974.

[23] A. Ehrenfeucht and H.P. Zeiger. Complexity measures for regular expressions. *J. Comput. Syst. Sci.*, 12(2) :134–146, 1976.

[24] V. M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16 :1–53, 1961.

[25] H. Gruber and J. Johannsen. Optimal lower bounds on regular expression size using communication complexity. In Roberto M. Amadio, editor, *FoSSaCS*, volume 4962 of *Lecture Notes in Computer Science*, pages 273–286. Springer, 2008.

[26] Y.-S. Han and D. Wood. Obtaining shorter regular expressions from finite-state automata. *Theor. Comput. Sci.*, 370(1-3) :110–120, 2007.

[27] J. E. Hopcroft. An $n \log(n)$ algorithm for minimizing the states in a finite automaton. In Z. Kohavi, editor, *The theory of machines and computations*, pages 189–196. Academic Press, New York, 1971.

[28] L. Ilie and S. Yu. Follow automata. *Inf. Comput.*, 186(1) :140–162, 2003.

[29] T. Jiang and B. Ravikumar. Minimal nfa problems are hard. *SIAM J. Comput.*, 22(6) :1117–1141, 1993.

[30] S. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, Ann. Math. Studies 34 :3–41, 1956. Princeton U. Press.

[31] D. Kozen. A completeness theorem for kleene algebras and the algebra of regular events. In *LICS*, pages 214–225. IEEE Computer Society, 1991.

[32] D. Kozen. A completeness theorem for kleene algebras and the algebra of regular events. *Inf. Comput.*, 110(2) :366–390, 1994.

[33] S. Lombardy and J. Sakarovitch. Derivatives of rational expressions with multiplicity. *Theor. Comput. Sci.*, 332(1-3) :141–177, 2005.

[34] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bull. Math. Biophysics*, 5 :115–133, 1943.

[35] R. F. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9 :39–57, March 1960.

[36] B. G. Mirkin. An algorithm for constructing a base in a language of regular expressions. *Engineering Cybernetics*, 5 :110–116, 1966.

[37] E. F. Moore. Gedanken experiments on sequential machines. In *Automata studies*, pages 129–153. Princeton Univ. Press, Princeton, N.J., 1956.

[38] A. Nerode. Linear automata transformation. In *Proceedings of AMS*, volume 9, pages 541–544. American Mathematical Society, 1958.

[39] C. Nicaud. On the average size of Glushkov's automata. In *LATA*, volume 5457 of *Lecture Notes in Computer Science*, pages 626–637. Springer, 2009.

[40] J.-L. Ponty, D. Ziadi, and J.-M. Champarnaud. A new quadratic algorithm to convert a regular expression into an automaton. In Darrell R. Raymond, Derick Wood, and Sheng Yu, editors, *Workshop on Implementing Automata*, volume 1260 of *Lecture Notes in Computer Science*, pages 109–119. Springer, 1996.

[41] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM J. Res.*, 3(2) :115–125, 1959.

[42] V. N. Redko. On defining relations for the algebra of regular events. *Ukrain. Mat.*, 16 :120–126, 1964.

[43] J. Sakarovitch. The language, the expression, and the (small) automaton. In J. Farré, I.Litovsky, and S. Schmitz, editors, *CIAA*, volume 3845 of *Lecture Notes in Computer Science*, pages 15–30. Springer, 2005.

[44] A. Salomaa. Two complete axiom systems for the algebra of regular events. *J. ACM*, 13(1) :158–169, 1966.

[45] A. M. Turing. On computable numbers with an application to the entscheidungs-problem. *Proc. London Math. Soc.*, 2(42) :230–265, 1936.

[46] D. Ziadi. Regular expression for a language without empty word. *Theor. Comput. Sci.*, 163(1&2) :309–315, 1996.

Index