الجمـــهوريـــة الجـــزائريـــة الديـمقـراطيـــــــة الشـــعبيـــــة
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
وزارة التعـلـيـم الـعـالـــي و الـبـحـث الـعـلـمي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE
جـامـعـة عمّـــار ثـلـيـــجـــي بالأغـــواط
UNIVERSITE AMAR TELIDJI LAGHOUAT
كـلـيـــــة الـعـلـــــوم
FACULTE DES SCIENCES
DEPARTEMENT DE MATHEMATIQUES ET INFORMATIQUE

Mémoire de MASTER

Domain : Mathématiques et Informatique
Filière : Informatiques
Option : Systèmes d'Information et de Décision

Par:
BENSAHA ZINEB

THEME

Etude comparative sur les techniques de passage d'une expression rationnelle vers un automate d'arbres

---

Soutenu publiquement le 08-06-2017 devant le jury composé de:

Mme. Cherroun *Hadda*	*Professeur*	*Président*
Mr. Lakhdar Kechna	*M.A(A)*	*Examinateur*
Mr. Guellouma Younes	*M.A(A)*	*Encadreur*

Année Universitaire 2016/2017

Dédécaces

Je dédie affectueusement ce modeste travail:

Á mes chérs parents:

- ***Á ma chére mère*** qui a uvr pour ma réussite, de par son amour, son soutien, tous les sacrifices consentis et ses précieux conseils, pour toute son assistance et sa présence dans ma vie, reçois à travers ce travail aussi modeste soit-il, l'expression de mes sentiments et de mon éternelle gratitude.

- ***A mon cher père*** qui peut étre fier et trouver ici le résultat de longues années de sacrifices et de privations pour m'aider à avancer dans la vie. Puisse Dieu faire en sorte que ce travail porte son fruit ; Merci pour les valeurs nobles, l'éducation et le soutient permanent venu de toi.

A mes frères et seours qui n'ont cessé d'étre pour moi des exemples de persévérance, de courage et de générosité.

A mes amis et spécialement à **Amina** et **Mimouna**.

A tous mes enseignants *qui ne m'ont guère privé de leur savoir et de leur bienséance.*

A mes collègues de la promotion .

Enfin à tous qui portent le nom ***BENSAHA*** *et à tous ceux qui connaissent.*

Remerciements

𝕿oute textit notre gratitude et remerciements au bon **DIEU**
qui nous a donné la force, le courage et la volonté délaborer ce travail.

Nous adressons notre profond remerciement à Mr ***Mr.GUELLOUMA Younes*** *Maitre assistant à l'université de Laghouat pour avoir dirigé ce travail avec tant de bienveillance. Nous le remercions pour ses précieux conseils et sa disponibilité.*

Nous adressons particulièrement nos remerciements aux membres du Jury.

Nous remercions tous le personnel de l'université de Amar TELIJI de Laghouat , l'université qui nous a accueilli bras ouvert , nos remerciements vont particulièrement aux enseignants et administrateurs du département de mathématique et d'informatique.

Nos remerciements les plus sincères à toutes les personnes qui nous ont aidé de prés ou de loin accomplir notre travail.

Enfin, nous exprimons nos vifs remerciements à toute notre famille et spécialement mes parents .

Bensaha Zineb

مُلخّص

الهدف من هذا العمل هو إجراء دراسة للتقنيات معبر القائم بين التعابير العادية ولدن شجرة. في الواقع، تعبير عادي هو تمثيل نصية من مشغلي تشكيلها. ويستخدم هذا الأخير لوصف نمط شجرة معينة. أما بالنسبة الآلي والأشجار، وأنها تعمل على التعرف على شجرة العائلة التالية قدر من الانتظام.

قبل تعميم نظرية لغات كليين يعرف من الكلمات، وهو التكافؤ بين اللغات العادية (وصفها مع تعبير عادي) ولغات معترف بها (المعترف بها من قبل إنسان شجرة) موجود. وحددت ثلاثة خوارزميات في الأدب، وهي طومسون الآلي والمواقف والمعادلات. والغرض من هذه الدراسة هو إجراء مقارنة من التعقيدات والتكافؤ بين هذه التقنيات الثلاث.

Résumé

Le but de ce travail est de faire une étude sur les techniques existante de passage entre les expressions régulières et les automates d'arbres. En effet, une expression rationnelle est une représentation textuelle formée d'opérateurs. Cette dernière sert à décrire un motif d'arbres donné. Quant aux automates d'arbres, ils servent à reconnaitre une famille d'arbre suivant une certaine régularité.

En généralisant le théorème de Kleene connu pour les langages de mots, une équivalence entre les langages réguliers (décrits avec une expression régulière) et des langages reconnaissables (reconnus par un automate d'arbre) est présente.

Trois algorithmes sont définis dans la littérature à savoir les automates de Thompson, de positions et des équations. Le but de cette étude est la d'effectuer une comparaison de complexités et d'équivalence entre ces trois techniques.

mots-clés: *automates d'arbres, expression régulière , Thompson, de positions des équations ,complexités*

Abstract

The aim of this work is to make a study on the existing techniques of passage between regular expressions and automata of trees. Indeed, a regular expression is a textual representation composed of operators. The latter is used to describe a given tree pattern. As for the automata of trees, they serve to recognize a family of trees according to a certain regularity.

By generalizing the Kleene theorem known for the language of words an equivalence between regular languages (described with a regular expression) and recognizable languages (recognized by a tree automaton) is present. Three algorithms are defined in the literature namely Thompson automata, positions and equations. The aim of this study is to perform a comparison of complexities and equivalence between these three techniques.

keywords: automata of trees,regular expression,Thompson automata, positions , equations , complexities.

Table des Matières

Liste des figures

Liste des tables

Introduction générale

Durant les dernières années, les chercheurs et les industriels se sont penché vers la théorie des langages d'arbres et des automates d'arbres afin de solutionner des problèmes complexes. Cette nouvelle tendance se matérialise dans plusieurs domaines comme dans les données XML et XML schémas [1, 2], le traitement du langage naturel [3], la vérification formelle et l'analyse des programmes. Conçu pour la première fois dans les années soixante [4], ce formalisme avait pour but de donner une généralisation de la théorie des langages pour les arbres faiblement utilisés dans des domaines concrets. En effet, la seule application majeure était la vérification des circuits intégrés. Durant les années 70, beaucoup de résultats théoriques concernant les automates d'arbres ont été établis. Mais un écart entre cette théorie en développement et la pratique a commencé à se manifester. En particulier, la complexité croissante des solutions données vis-à-vis du problème de "décidabilité" et l'incohérence entre ses résultats et les ressources matérielles de l'époque. Petit à petit, cette théorie a repris sa place dans la pratique. Le retour considérable vers la structure arborescente a suscité de rediscuter, améliorer et appliquer ce genre d'outils formels qui sont les automates d'arbres.

Automates d'arbres Les automates d'arbres sont une généralisation des automates de mots. Ils sont parfaitement adaptés pour la modélisation des structures de

données arborescentes (pages Web, XML, vérification des programmes, récursivité) et sont de plus en plus utilisés comme modèles de langages en traitement automatique des langages naturels. Les défis posés par l'analyse de grandes masses de données, en extraction automatique en particulier, ont suscité de nouveaux travaux sur l'inférence grammaticale de ces automates. Les langages réguliers de mots et les automates finis qui les reconnaissent sont des outils formels de base pour traiter les séquences de données.

Expressions rationnelles Une expression rationnelle d'arbres appelés aussi motifs d'arbres est aussi une généralisation de celle sur les mots. Elle est aussi une chaîne de caractère incluant des opérateurs fondamentaux permettant de décrire -selon une syntaxe précise- un ensemble d'arbres ou bien un langage d'arbres dit*rationnel*. De la même façon, les langages rationnels d'arbres permettent de manipuler les structures arborescentes de données telles que les pages Web.

Problématique Le but de ce travail est de faire une étude sur les techniques existante de passage entre les expressions régulières et les automates d'arbres. En effet, une expression rationnelle est une représentation textuelle formée d'opérateurs. Cette dernière sert à décrire un motif d'arbres donné. Quant aux automates d'arbres, ils servent à reconnaitre une famille d'arbre suivant une certaine régularité.
En généralisant le théorème de Kleene connu pour les langages de mots, une équivalence entre les langages réguliers (décrits avec une expression régulière) et des langages reconnaissables (reconnus par un automate d'arbre) est présente. Trois algorithmes sont définis dans la littérature à savoir les automates de Thompson, de positions et des équations. Le but de cette étude est la d'effectuer une comparaison de complexités et d'équivalence entre ces trois techniques. Un méta-algorithm

Plan de la thèse Omit cette introduction, ce mémoire de thèse est organisé en 3 chapitres.

- Le premier chapitre présente les définitions de notions et notations utiles pour la compréhension des arbres, langages d'arbres et expressions rationnelles qui dénotent la classe des langages réguliers.
- Le deuxième chapitre prend étudions des techniques existante pour la conversion d'une expression rationnelle en un automate équivalent.
- le troisième chapitre. c'est le dernier chapitre, et dans le j'ai comparé entre les différents construction d'automates.
- Finalement, nous présentons nos conclusions

Chapitre 1

Généralités sur les arbres

1 Introduction

Dans ce chapitre , nous présentons la Notions et définitions sur les arbres,les opération sur les arbres ,les automates par les mots et par les arbres , expression régulière par les mots et par les arbres et langages d'arbres

2 Notions et définitions sur les arbres

Dans cette section, nous allons introduire les notions et les définitions nécessaires pour la bonne compréhension du reste du mémoire.

Omit nos propres définitions, la plupart des notions, notations et définitions présentées dans ce chapitre sont principalement compilées à partir du document de référence "tata" [5]. D'autres référence sont aussi utilisées comme [6, 7, 8].

2.1 Alphabet, mots et langages

Afin de faire l'analogie avec la théorie des langages, nous commençons par rappeler quelques notions basiques mais nécessaire pour normaliser les formalismes [7, 8].
Un alphabet est un ensemble fini de symboles. Un symbole est appelé "lettre".
Un mot est une suite finie (éventuellement vide) de lettres.
Un arbre est une collection (éventuellement vide) de noeuds et d'arêtes assujettis à certaines conditions : un noeud peut porter un nom et un certain nombre d'informations pertinentes ; une arête est un lien entre deux noeuds. Plus de détails concernant la structure d'arbre, la définition formelle, les fonctions ainsi que des exemples sont présentés ci-dessous
Le mot vide est noté par convention ε.
L'étoile de Kleene Σ^* (resp. Σ^+) est l'ensemble de tous les mots (resp. l'ensemble

de tous les mots non-vides) qu'on peut construire à partir de l'alphabet Σ. En effet, l'étoile de Kleene peut être construite comme suit :

- $\varepsilon \in \Sigma^*$ est un symbole spécial appelé le mot vide,
- $\sigma \subset \Sigma^*$ pour tout $\sigma \in \Sigma$,
- si $u, v \in \Sigma^*$ alors $uv, vu \in \Sigma^*$.

La longueur d'un mot u notée $|u|$ correspond au nombre total des lettres de u. Par conséquent, $|\ |$ est une application de $\mathbb{N} \rightarrow \sigma^*$ qui est définie par induction comme suit:

- $|\varepsilon| = 0$,
- $\forall a \in \sigma, |a| = 1$,
- Soient $u, v \in \sigma^*$ alors $|uv| = |u| + |v|$.

L'opération de concaténation : de deux mots $u, v \in \Sigma^*$ résulte un nouveau mot uv constitué par la juxtaposition de lettres de u et de lettres de v. Nous avons alors $|uv| = |u| + |v|$. La concaténation est une opération interne de Σ^*; elle est associative, mais pas commutative (sauf dans le cas dégénéré où Σ ne contient qu'un seul symbole). ε est l'élément neutre pour la concaténation : $u\varepsilon = \varepsilon u = u$. Si u se factorise sous la forme $u = xy$, alors on écrira $y = x^{-1}u$. La position : Pos^u d'un nombre entier $i \in \mathbb{N}$ est une application $\mathbb{N} \rightarrow \sigma$ qui retourne le symbole se trouvant à la $i-$éme position d'un mot u.

Un langage est un sous ensemble de Σ^* pour un alphabet Σ donné. Un langage peut être fini ou infini.

Soit $\Sigma = \{a, b, c, d\}$. Alors :

- $\Sigma^* = \{\varepsilon, a, b, c, ab, ba, ac, ca, abc, acb, \ldots\}$, soit l'ensemble de tous les mots composés de lettre de Σ.
- $\{\varepsilon, a, b, c, d\}, \{a, ab, abc, abcd\}$ sont deux langages finis.
- $\{a, aa, aaa, aaaa, \ldots\}$ est un langage infini qui contient tous les mots composés à partir de la lettre a.
- $|\varepsilon| = 0, |abcd| = |aaaa| = 4$
- $Pos^{abcd}(2) = b$

3 Opérations sur les arbres

Il existe plusieurs opérations utiles sur les arbres. Deux parmi elles ont déjà été citées. Il s'agit de l'opération $\mathcal{R}$ qui retourne le symbole de la racine d'un arbre et St qui retourne l'ensemble des sous arbres. Dans cette section, nous allons présenter d'autres opérations très intéressantes : substitution, concaténation.

3.1 La substitution

La substitution peut être vue comme un remplacement d'un sous arbre d'un arbre cible par un autre arbre donné. Selon [6], il en existe trois types de substitutions.

Définition 1.1 *(Substitution d'une seule occurrence d'un sous arbre) Soit* $s, t \in T_\Sigma$. $t[\overset{d}{\leftarrow} s]$ *dénote l'arbre dans lequel l'arbre* $u = Pos^t(d)$ *est remplacé par* s.

Définition 1.2 *(Substitution de toutes les occurrences d'un sous arbres) Soient* $s, t \in T_\Sigma$. $t[u \leftarrow s]$ *dénote l'arbre dans lequel toutes les occurrences de l'arbre* u *sont remplacées par l'arbre* s. *Cette substitution est définie comme suit :*

- $u[u \leftarrow s] = s$,

- $t[u \leftarrow s] = t$ *si* $s \notin St(t)$,
- $f(t_1,\ldots,t_{\hat{f}})[u \leftarrow s] = f(t_1[u \leftarrow s],\ldots,t_{\hat{f}}[u \leftarrow s])$.

Concaténation d'arbres

Étant un cas spécial de la substitution, la concaténation d'arbres est une opération très utilisée car elle participe dans la définition d'une classe de régularité pour les arbres [9]. Il s'agit d'une substitution ou l'arbre à substituer est une constante.

Définition 1.3 *(Concaténation d'arbres) Soit* $s,t \in T_\Sigma$. $t._c s$ *dénote l'arbre où toutes les occurrences de* $c \in \Sigma_0$ *sont remplacées par l'arbre s.*

- $c._c s = s$,
- $d._c s = d$ *si* $d \neq c$,
- $f(t_1,\ldots,t_{\hat{f}})._c s = f(t_1._c s,\ldots,t_{\hat{f}}._c s)$.

La concaténation est une loi de composition interne dans T_Σ satisfaisant :

- Elle n'est pas commutative : $t._c u \neq u._c t$,
- Elle est associative : $s._c(t._c u) = (s._c t)._c u$,
- L'élément neutre de $._c$ est c.

4 Les automates

wikipedia "Un automate fini ou automate avec un nombre fini d'états (en anglais finite-state automaton ou finite state machine) est un modèle mathématique de calcul, utilisé dans de nombreuses circonstances, allant de la conception de programmes informatiques et de circuits en logique séquentielle aux applications

dans des protocoles de communication, le contrôle des processus, la linguistique et même la biologie [10]

4.1 Les automates pour les mots

Définition 1.4 *(Automate fini). Un automate fini sur un alphabet* Σ *est un tuple* $A = (\Sigma, Q, I, F, \delta)$ Q *est un ensemble fini d' états,*$I \subseteq Q$ *l' ensemble des états initiaux,* $F \subseteq Q$ *l' ensemble des états finals, et* $\delta \subseteq Q \times \Sigma \cup \{\epsilon\} \times Q$ *la relation de transition. Alternativement, on peut voir* δ *comme une fonction de* $Q \times \Sigma \cup \{\epsilon\}$ *dans* 2^Q *l' ensemble des parties de* Q*. On étend naturellement la définition de* δ *sur* $Q \times \Sigma^* \times Q$ *par* $\delta^*(q, \epsilon) = q$ *et* $\delta^*(q, aw) = \cup_{q' \in \delta(q,a)} \delta(q', w)$*. Par un abus de notation bien pratique, on peut aussi l'étendre des ensembles d' états ou de mots par l' union* $\delta^*(E, L) = \cup_{q \in E, u \in L} \delta^*(q, u)$*. Enfin, on peut définir similairement la relation inverse* $\delta^{-1} = \{(q, a, p) \mid (p, a, q) \in \delta\}$.

4.2 Les automates pour les arbres

Dans la théorie des langages réguliers des mots, les automates finis sont la représentation la plus utilisée. Dans cette théorie, les automates déterministes et non-déterministes sont considérés comme équivalents de point de vue d'acceptation. La direction de traitement des mots est sans importance, Plusieurs types d'automates finis d'arbre, qui sont d'ailleurs une généralisation des automates de mots, peuvent être définis. Ici, la direction de traitement de l'arbre est significative. Les automates sont divisés en plusieurs catégories selon deux critères : déterminisme et direction de traitement. Il y a des automates déterministes et non-déterministes, qui peuvent à leurs tours être descendants ou ascendant . Les automates déterministes descendants sont moins puissants que les autres qui sont

tous équivalents. Un automate d'arbre est composé d'un ensemble d'états, un ensemble de transitions, un ou plusieurs états finaux et un alphabet. L'automate a un seul état final s'il est ascendant et plusieurs s'il est descendant. Contrairement aux automates reconnaissant les mots, où la transition s'exprime par une relation entre deux états, dans les automates d'arbre, une transition est une fonction affectant d'une part un état et d'autre part un ensemble d'états. Ceci s'explique par l'arité des symboles de l'alphabet. Plusieurs définitions formelles d'automates d'arbre existent dans la littérature. Elles sont similaires et utilisent un alphabet à rang borné.[11]

Définition 1.5 *Un automate d'arbre A est un quadruplet* (Q, Σ, Δ, Q_T) *tel que:*
.Q *est un ensemble fini d'états,*
. Σ *est un alphabet a rang borne,*
. $\Delta = \{\delta_a, a \in \Sigma\} \cup \{\delta_\epsilon\}$ *est l'ensemble des relations de transition, ou* $\delta_a \subseteq Q \times Q_n$ *pour tout* $a \in \Sigma$ *et* $\delta_\epsilon \subseteq Q \times Q$ *est la relation de transition vide, et*
. $Q_T \times Q$ *est l'ensemble des etats finaux. Les relations de transition sont de la forme*
$f(q_1(x_1), q_2(x_2), ..., q_n(x_n)) \longrightarrow q(f(x_1, x_2, ..., x_n)), oun \geq 0, f \in \Sigma_n, q, q_1, ..., q_n \in Qetx_1, x_2, ..., x_n \in X.$
Si le symbole est une constante, la relation de transition est de la forme $a \longrightarrow q(a)$*. Ces regles sont considerees comme des regles initiales (avec lesquelles nous commencons les derivations).*

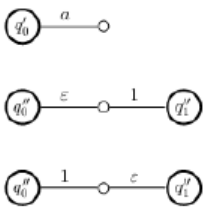


Figure 1.1: Exemples de transitions simples dans un automate d'arbre

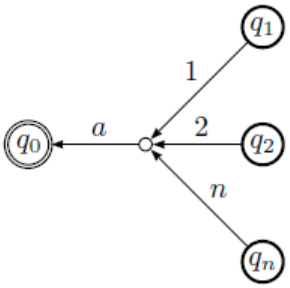


Figure 1.2: Exemple de transition multiple dans un automate d'arbre

5 Expression régulière

"une expression régulière ou expression normale ou expression rationnelle ou motif, est une chaîne de caractères, qui décrit, selon une syntaxe précise, un ensemble de chaînes de caractères possibles."

5.1 Expression pour les mots

Langages reconnaissables

Définition 1.6 *(Langage reconnaissable). Le langage reconnu par un automate est* $A = (\Sigma, Q, I, F, \delta)$

$$L(A) = \{w \in \Sigma^* \mid \exists i \in I, f \in F, f \in \delta^*(i, w)\}$$

Deux automates finis sur Σ sont équivalents s' ils reconnaissent le m?me langage de Σ^*. Un langage L sur Σ^* est reconnaissable s' il existe un automate fini sur Σ tel que $L = L(A)$. L' ensemble des langages reconnaissables sur Σ^* est noté Rec(Σ^*).

Langages rationnels

Définition 1.7 *(Langage rationnel) Rat(Σ^*) est le plus petit ensemble de parties de Σ^* qui contient les parties finies de Σ^* et qui est fermé par union, concaténation et étoile. Un langage rationnel sur Σ^* est un élément de Rat(Σ^*). Comme chaque partie finie de Σ^* est soit l' ensemble vide, soit une union finie de mots de Σ^* , et que ces mots sont soit le mot vide, soit la concaténation finie de lettres de Σ, on obtient une définition équivalente sous la forme d' expressions rationnelles.*

Définition 1.8 *(Expression rationnelle) Une expression rationnelle E sur Σ est un terme défini inductivement par*

$$E ::= \emptyset \mid \epsilon \mid a \mid E_1^* \mid E_1 + E_2 \mid E_1 E_2$$

ou a est un symbole de Σ, et E_1 et E_2 sont des expressions rationnelles. On interpr?te une expression rationnelle E comme un langage L(E) sur Σ^* défini inductivement par

$$L(\emptyset) = \emptyset$$

$$L(\epsilon) = \{\epsilon\}$$

$$L(a) = \{a\}$$

$$L(E^*) = L(E)^*$$

$$L(E_1 + E_2) = L(E_1)[L(E_2)$$

$$L(E_1E_2) = L(E_1)L(E_2)$$

Le langage d' une expression rationnelle est bien un langage rationnel. Deux mesures sont couramment utilisées pour la taille d' une expression rationnelle E : $|\ E\ |$ *la taille du terme E, définie par*

$$|\ \epsilon\ | = |\ \epsilon\ | = |\ a\ | = 1$$

$$|\ E^*\ | = |\ E\ | + 1$$

$$|\ E_1 + E_2\ | = |\ E_1E_2\ | = |\ E_1\ | + |\ E_2\ | + 1$$

L' autre mesure est le nombre d' occurences de symboles alphabétiques dans E, défini par

$$\|\ \epsilon\ \| = \|\ \epsilon\ \| = 0$$

$$\|\ a\ \| = 1$$

$$\|\ E^*\ \| = \|\ E\ \|$$

$$\|\ E_1 + E_2\ \| = \|\ E_1E_2\ \| = \|\ E_1\ \| + \|\ E_2\ \|$$

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

Théorame: 1.9 *(Kleene, [12]) Soit* Σ *un alphabet. Alors :* $Rec(\Sigma^*) = Rat(\Sigma^*)$. *Démonstration. Les méthodes constructives permettant de démontrer les deux sens de l'inclusion .* $Rat(\Sigma^*) \subseteq Rec(\Sigma^*)$

5.2 Expression pour les arbres

Les expressions régulières d'arbre sont des opérateurs permettant de décrire les langages réguliers d'arbre définis sur un mémé alphabet Σ. Contrairement aux mots , les arbres peuvent être concaténés en plusieurs endroits, les mots étant connectes seulement a la fin.

Construction d'expressions régulières d'arbre Nous définissons l'ensemble $Ret(\Sigma)$ des expressions régulières sur l'alphabet Σ.

1. l'ensemble vide $\emptyset$ est une expression régulière,
2. Si $a \in \Sigma_0$, alors $a \in Ret(\Sigma)$,
3. Si $f \in \Sigma_n$ et $E_1, E_2, ..., E_n \in Ret(\Sigma)$ alors $f(E_1, E_2, ..., E_n) \in Ret(\Sigma)$,
4. Si $E_1, E_2 \in Ret(\Sigma)$ alors $(E_1 + E_2) \in Ret(\Sigma)$,
5. Si $E_1, E_2 \in Ret(\Sigma)$ et $c \in \Sigma_0$, alors $(E_1 ._c E_2) \in Ret(\Sigma \cup \{\epsilon\})$,
6. Si $E \in Ret(\Sigma \cup \{\epsilon\})$ et $c \in \Sigma_0$, alors $E^{*c} \in (\Sigma \cup epsilon\})$.

Définition 1.10 *Une* Expression Rationnelle d'Arbres (ERA) E *à travers un alphabet* Σ *est un terme défini par induction comme suit :*

$$E = 0 \tag{1.1}$$

$$E = f(E_1, \ldots, E_n) \tag{1.2}$$

$$E = E_1 + E_2 \tag{1.3}$$

$$E = E_1 \cdot_c E_2 \tag{1.4}$$

$$E = E_1^{*_c} \tag{1.5}$$

avec $f \in \Sigma_n$, $c \in \Sigma_0$ *et* E_i *est une ERA pour* $i = 1 \ldots n$.

Afin d'extraire la sémantique d'une ERA (le langage dénoté par une ERA) nous introduisons la définition suivante :

Définition 1.11 *La* sémantique *d'une ERA E est le langage d'arbres $[[E]]$ défini par induction comme suit :*

$$[[0]] = \emptyset \tag{1.6}$$

$$[[a]] = \{a\} \tag{1.7}$$

$$[[E_1 + E_2]] = [[E_1]] \cup [[E_2]] \tag{1.8}$$

$$[[E_1 \cdot_c E_2]] = [[E_1]] \cdot_c [[E_2]] \tag{1.9}$$

$$[[E^{*c}]] = [[E]]^{*c} \tag{1.10}$$

$$[[f(E_1, \ldots, E_n)]] = \{f(s_1, \ldots, s_n) \mid s_1 \in [[E_1]], \ldots, s_n \in [[E_n]]\} \tag{1.11}$$

ou $f \in \Sigma_n$, $c \in \Sigma_0$ et E_i est une ERA pour $i = 1 \ldots n$.

On dit que deux ERA E et F sont *équivalentes* ($E \equiv F$) si et seulement si $[[E]] = [[F]]$.

6 Langages d'arbres

La notion de langage d'arbres est très utile dans le traitement des grandes masses d'informations car elle donne la possibilité de décrire/reconnaître des familles d'arbres qui ont des caractéristiques spécifiques. Dans cette section, nous nous intéressons aux fondements formels des langages d'arbres.

Définition 1.12 *Soit T_Σ l'ensemble des arbres rangés définis sur Σ. Un langage d'arbres L est un sous-ensemble de T_Σ. ($L \subset T_\Sigma$).*

Vue que les langages d'arbres sont des ensembles, on peut utiliser les opérations usuelles tel que l'union de deux langages d'arbres K et L ($K \cup L$), l'intersection ($K \cap L$), le complément $\complement(L)$...En raison de simplicité, nous définissons quelques notations nécessaires pour la lecture du reste du mémoire.

Opérations sur les langages d'arbres

Les mêmes opérations de substitution sur les arbres peuvent être étendue sur les langages d'arbres

Définition 1.13 *Soit,* $f \in \Sigma_n, L, L_1, \ldots, L_n \in T_\Sigma$.

- $f(L_1, \ldots, L_n) = \{f(t_1, \ldots, t_n) \mid t_i \in L_i, i = 1, n\}$ *(forêt de langages),*
- $L[\overset{d}{\leftarrow} s] = \{t[\overset{d}{\leftarrow} s] \mid t \in L\}$*(Substitution d'une seule occurrence d'un sous arbre dans un langage),*
- $L[u \leftarrow s] = \{t[u \leftarrow s] \mid t \in L\}$ *(substitution d'un arbre dans un langage),*
- $L_1 ._c L_2 = \{t_1 ._c t_2 \mid t_1 \in L_1, t_2 \in L_2\}$ *(concaténation de deux langages ou produit-c).*

Le produit-c est aussi connu sous le nom de *out-inside* (OI) substitution [13]. Nous rappelons quelques propriétés algébriques liées à cette notion (voir [13, 14, 15]). Comme dans le cas des mots, le produit-c est distributif sur l'union :

Lemme 1.14 *Soient* L_1, L_2 *et* L_3 *trois langages d'arbres à travers* Σ. *Soit* c *un symbole dans* Σ_0. *Alors:*

$$(L_1 \cup L_2) \cdot_c L_3 = (L_1 \cdot_c L_3) \cup (L_2 \cdot_c L_3)$$

Soit t un arbre dans T_Σ. Alors:

$$\begin{aligned} t \in (L_1 \cup L_2) \cdot_c L_3 &\Leftrightarrow \exists u \in L_1 \cup L_2, \exists v \in L_3, t = u \cdot_c v \\ &\Leftrightarrow (\exists u \in L_1, \exists v \in L_3, t = u \cdot_c v) \vee (\exists u \in L_2, \exists v \in L_3, t = u \cdot_c v) \\ &\Leftrightarrow t \in (L_1 \cdot_c L_3) \cup (L_2 \cdot_c L_3) \end{aligned}$$

Une autre propriété commune est l'associativité.

Lemme 1.15 *Soient t et t' deux arbres dans T_Σ. Soit L un langage d'arbres à travers Σ et soit c un symbole dans Σ_0. Alors:*

$$t \cdot_c (t' \cdot_c L) = (t \cdot_c t') \cdot_c L$$

Par induction sur la structure de t.

1. $t = c$.Alors $t \cdot_c (t' \cdot_c L) = t' \cdot_c L = (t \cdot_c t') \cdot_c L$.
2. $t \in \Sigma_0 \setminus \{c\}$.Alors $t \cdot_c (t' \cdot_c L) = t = (t \cdot_c t') \cdot_c L$.
3. $t = f(t_1, \ldots, t_n)$ avec $n > 0$. Alors :

$$\begin{aligned} f(t_1, \ldots, t_n) \cdot_c (t' \cdot_c L) &= f(t_1 \cdot_c (t' \cdot_c L), \ldots, t_n \cdot_c (t' \cdot_c L)) \\ &= f((t_1 \cdot_c t') \cdot_c L, \ldots, (t_n \cdot_c t') \cdot_c L) \quad \text{(Hypothèse d'induction)} \\ &= f(t_1 \cdot_c t', \ldots, t_n \cdot_c t') \cdot_c L \\ &= (f(t_1, \ldots, t_n) \cdot_c t') \cdot_c L \end{aligned}$$

Corollaire 1.16 *Soient L, L' et L'' trois langages à travers Σ, $c \in \Sigma_0$. Alors :*

$$L \cdot_c (L' \cdot_c L'') = (L \cdot_c L') \cdot_c L''$$

Ainsi, nous pouvons conclure que le produit-c est compatible avec l'inclusion.

Lemme 1.17 *Soit t un arbre à travers Σ, et L, L′ deux langages de T_Σ tel que $L \subset L'$. Alors:*

$$t \cdot_c L \subset t \cdot_c L'$$

Par conséquent :

Corollaire 1.18 *Soient L, $L' \subset L''$ trois langages T_Σ et $c \in \Sigma_0$. Alors :*

$$L \cdot_c L' \subset L \cdot_c L''$$

Une nouvelle propriété qui diffère du cas des mots est présentée dans le lemme suivant.

Lemme 1.19 *Soient t_1, t_2 et t_3 trois arbres de T_Σ. Soient a et b deux symboles distincts dans Σ_0 tel que a n'apparaît pas dans t_3. Alors:*

$$(t_1 \cdot_a t_2) \cdot_b t_3 = (t_1 \cdot_b t_3) \cdot_a (t_2 \cdot_b t_3)$$

Par induction sur t_1.

1. Si $t_1 = a$, alors

$$(t_1 \cdot_a t_2) \cdot_b t_3 = t_2 \cdot_b t_3 = (t_1 \cdot_b t_3) \cdot_a (t_2 \cdot_b t_3)$$

2. Si $t_1 = b$, alors

$$(t_1 \cdot_a t_2) \cdot_b t_3 = t_3 = (t_1 \cdot_b t_3) \cdot_a (t_2 \cdot_b t_3)$$

3. Si $t_1 = c \in \Sigma_0 \setminus \{a, b\}$, alors

$$(t_1 \cdot_a t_2) \cdot_b t_3 = t_1 = (t_1 \cdot_b t_3) \cdot_a (t_2 \cdot_b t_3)$$

4. Si $t_1 = f(u_1,\dots,u_n)$ avec $n > 0$, Alors

$$\begin{aligned}(t_1 \cdot_a t_2) \cdot_b t_3 &= (f(u_1 \cdot_a t_2,\dots,u_n \cdot_a t_2)) \cdot_b t_3 \\ &= f((u_1 \cdot_a t_2) \cdot_b t_3,\dots,(u_n \cdot_a t_2) \cdot_b t_3) \\ &= f((u_1 \cdot_b t_3) \cdot_a (t_2 \cdot_b t_3),\dots,(u_n \cdot_b t_3) \cdot_a (t_2 \cdot_b t_3)) \quad \text{(Hypothése d'induction)} \\ &= f(u_1 \cdot_b t_3,\dots,u_n \cdot_b t_3) \cdot_a (t_2 \cdot_b t_3) \\ &= (f(u_1,\dots,u_n) \cdot_b t_3) \cdot_a (t_2 \cdot_b t_3)\end{aligned}$$

7 Conclusion

Ce chapitre est une introduction aux langages réguliers d'arbre. Nous y avons présentés les notions de base de ce type de langages ainsi que ses différentes représentations:les Opération sur les arbres, les automates, les expressions régulières pour les mots et les arbres et langage d'arbres.Ces notions seront utilisées dans les chapitres suivants pour l'étude les automates de Thompson,Position et Équation .

Chapitre 2

Conversion d'une ER ver un automate

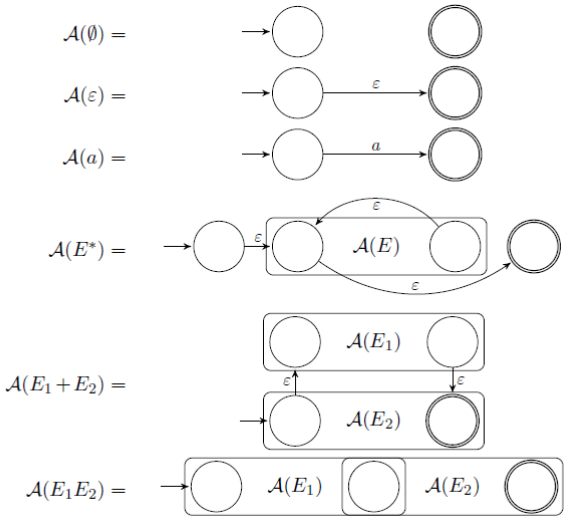
1 Introduction

Dans ce chapitre, nous étudions les techniques existante pour la conversion d'une expression rationnelle en un automate équivalent.

2 Conversion dans les mots

2.1 Automate de Thompson

La première méthode présentée est l'automate de Thompson qui est une construction simple et intuitive. :[16] on construit un automate A(E) par induction sur l' expression rationnelle E. Les automates généralisé vérifient comme invariant qu'ils ont un unique état initial, qui n a pas de transition entrante et un unique état final, qui n a pas de transition sortante. Cet invariant permet d'assurer la validité de la construction pour la concaténation et l'étoile.[17]



Cette algorithme inductif travaille en $O(\mid E \mid)$ sur la taille de l'expression, et résulte en un automate avec ϵ-transitions avec au plus $2 \mid E \mid$ états et $3 \mid E \mid$ transitions, soit au final $O(\mid E \mid^2)$ une fois les ϵ-transitions éliminées.

Exemple 2.1 *: si on considéré l'expression* $E = (ab + b)^*ba$ *, on obtient par cette construction l'automate*

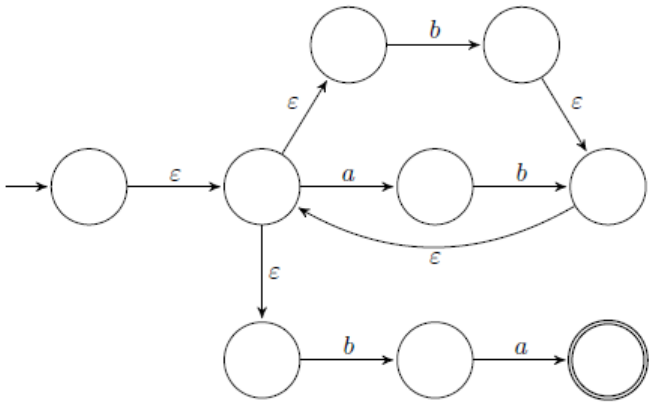


Figure 2.1: Exemple de l'automate Thompson exemple(2.1)

2.2 L' Automate de Glushkov

La deuxième méthode étudiée est la méthode de Glushkov [18] (méthode définie indépendamment par McNaughton et Yamada dans [19]), permettant de calculer depuis une expression à n lettres un automate $(n + 1)$ 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 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 $Pos(E)$ explicite les différentes positions de l'expression linéarisée de E. La fonction $Null(E)$ retourne le singleton ϵ si le mot vide est contenu dans le langage dénote par E, l'ensemble vide $\emptyset$ sinon.

La fonction $First(E)$ retourne les positions des lettres pouvant débuter un mot du langage. La fonction $Last(E)$ retourne les positions des lettres pouvant terminer un mot du langage. Enfin, la fonction $Follow(E,x)$, où x est un element de $Pos(E)$, retourne les positions pouvant succeder à la lettre en position x dans un mot du langage dénote par l' expression E. Soit E une expression rationnelle, E^* son expression linéarisée et x un élément de Σ_{E^*} . Les cinq ensembles sont définis formellement de la façon suivante :

Définition 2.2 *(Définitions des Fonctions de Glushkov)*

$$
\begin{aligned}
Pos(E) &= \Sigma_{E^*} \\
Null(E) &= \begin{cases} \{\epsilon\} & si \epsilon \in L(E), \\ \emptyset & sinon. \end{cases} \\
First(E) &= x \in Pos(E) \mid \exists w' \in \Sigma^*_{E^*}, x, w' \in L(E^* \\
Last(E) &= x \in Pos(E) \mid \exists w' \in \Sigma^*_{E^*}, w', x \in L(E^*) \\
Follow(E,x) &= x_0 \in Pos(E) \mid \exists w', w'' \in \Sigma^*_{E^*}, w', x, x', w'' \in L(E^*)
\end{aligned}
$$

Exemple 2.3 *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 :*

$$Pos(E) = \{1,2,3,4\} \qquad Follow(E,1) = \{1,2,3\}$$

$$Null(E) = \{\epsilon\} \qquad Follow(E,2) = \{1,2,3\}$$

$$First(E) = \{1,2,3,4\} \qquad Follow(E,3) = \emptyset$$

$$Last(E) = \{3, 4\} \qquad Follow(E, 4) = \{4\}$$

On pourra vérifier que les mots $h_E(\epsilon) = \epsilon, h_E(11213) = aabaa, h_E(3) = a, h_E(22123) = bbabaeth_E(44) = bb$ *appartiennent au langage L(E) selon les formules de la définition*

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

1):Calcul de la Fonction Pos.

$Pos(\emptyset)$	$=$	$\emptyset$	$Pos(F+G)$	$=$	$Pos(F) \cup Pos(G)$
$Pos(\epsilon)$	$=$	$\emptyset$	$Pos(F.G)$	$=$	$Pos(F) \cup Pos(G)$
$Pos(a)$	$=$	a	$Pos(F^*)$	$=$	$Pos(F)$

2:)Calcul de la Fonction Null.

$Null(\emptyset)$	$=$	$\emptyset$	$Null(F+G)$	$=$	$Null(F) \cup Null(G)$
$Null(\epsilon)$	$=$	ϵ	$Null(F.G)$	$=$	$Null(F) \cap Null(G)$
$Null(a)$	$=$	$\emptyset$	$Null(F^*)$	$=$	$Null(\epsilon)$

3):Calcul de la Fonction First.

$First(\emptyset)$	$=$	$\emptyset$	$First(F+G)$	$=$	
$First(\epsilon)$	$=$	$\emptyset$	$First(F.G)$	$=$	$\begin{cases} First(F) \cup First(G) & siNull(\\ First(F) & sinon. \end{cases}$

4):Calcul de la Fonction Last.

$$Last(\emptyset) = \emptyset \qquad Last(F+G) =$$

$$Last(\epsilon) = \emptyset \qquad Last(F.G) = \begin{cases} Last(F) \cup Last(G) & siNull(G) \\ Last(G) & sinon. \end{cases}$$

5): Calcul de la Fonction Follow.

$$Follow(\emptyset, x) = Follow(\epsilon, x) = Follow(a, x) = \emptyset$$

$$Follow(F+G, x) = Follow(F, x) \cup Follow(G, x)$$

$$Follow(F.G, x) = \begin{cases} Follow(F, x) \cup First(G) & six \in Last(F), \\ Follow(F, x) \cup Follow(G, x) & sinon. \end{cases}$$

$$Follow(F^*, x) = \begin{cases} Follow(F, x) \cup First(F) & six \in Last(F), \\ Follow(F, x) & sinon. \end{cases}$$

2.3 Expressions derivees partielles d' ANTIMIROV

La troisième méthode présentée est la dérivation . Cette technique de construction d'un automate équivalent à une expression rationnelle fournit des automates, généralement non déterministes, encore plus compacts que ceux de l'automate de GLUSHKOV. Cette construction est un raffinement de celle mise au point par BRZOZOWSKI [1964], et le déterminisé de l' automate que l'on obtient est exactement celui que l'on aurait obtenu par la construction de BRZOZOWSKI

Définition 2.4 *(Dérivée d'une expression). La dérivée partielle $\partial_a(E)$ d' une expression rationnelle E sur Σ par une lettre a de Σ est l' ensemble d' expressions rationnelles sur Σ défini par*

$$
\begin{aligned}
\partial_a(\emptyset) &= \emptyset \\
\partial_a(\epsilon) &= \emptyset \\
\partial_a(b) &= \begin{cases} \{\epsilon\} & si \quad a = b \\ \emptyset & sinon \end{cases} \\
\partial_a(E+F) &= \partial_a(E) \cup \partial_a(F) \\
\partial_a(E^*) &= \partial_a(E).\{E^*\} \\
\partial_a(EF) &= \begin{cases} \partial_a(E).\{F\} & si \quad \epsilon \notin L(E) \\ \partial_a(E).\{F\} \cup \partial_a(F) & sinon \end{cases}
\end{aligned}
$$

où l'opération de concaténation est étendue de maniére évidente aux ensembles d'expressions rationnelles. Attention, dans $\partial_a(\emptyset) = \emptyset$ *, le symbole* $\{\emptyset\}$ *du terme de gauche est une expression rationnelle, tandis que celui du terme de droite est l'ensemble vide ! On étend cette définition àdes mots* w *de* Σ^* *et à des ensembles d'expressions rationnelles* S *par*

$$
\begin{aligned}
\partial_\epsilon(E) &= \{E\} \\
\partial_{wa}(E) &= \partial_a(\partial_w(E)) \\
\partial_w(S) &= \cup_{E \in S} \partial_w(E)
\end{aligned}
$$

Exemple 2.5 *Par exemple , si on pose* $E = (ab + b)^*ba$ *, on obtient les dérivées partielles successives*

$$
\begin{aligned}
\partial_a(E) &= \{b(ab+b)^*ba\} \\
\partial_b(E) &= \{E, a\} \\
\partial_a(b(ab+b)^*ba) &= \emptyset \\
\partial_b(b(ab+b)^*ba) &= \{E\} \\
\partial_a(a) &= \{\epsilon\} \\
\partial_b(a) &= \emptyset \\
\partial_a(\epsilon) &= \emptyset \\
\partial_b(\epsilon) &= \emptyset.
\end{aligned}
$$

Cet ensemble de dérivées partielles se traduit aisément en un automate

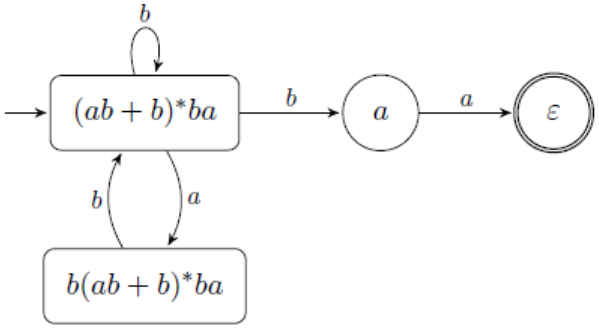


Figure 2.2: Automate de dérivées partielles de exemple 2.10

3 Conversion dans les arbres

Avant d'entamer notre contribution, nous présentons les techniques existantes pour transformer une ER en un AAFA á savoir l'automate de Thompson, de position et d'équations.

3.1 Automate de Thompson

Récemment, Belabbaci [11] a proposé une généralisation de l'automate de Thompson pour les mots [16] aux arbres. En effet, l'automate de Thompson pour les arbres est aussi un automate non déterministe avec ϵ-transitions. Pour sa construction, une forme normale a été conçue (voir Figure 2.3). Plus précisément,

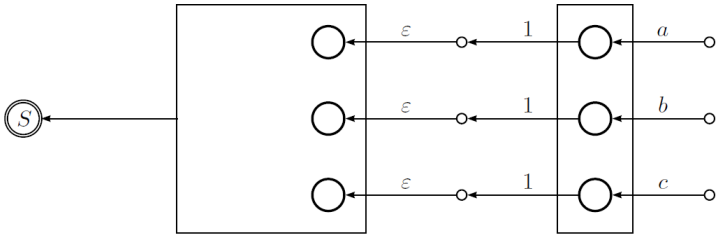
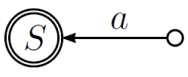


Figure 2.3: Forme générale de l'automate de Thompson

l'automate de Thompson $\mathcal{T}_E$ de l'ER E est construit d'une manière inductive comme suit :

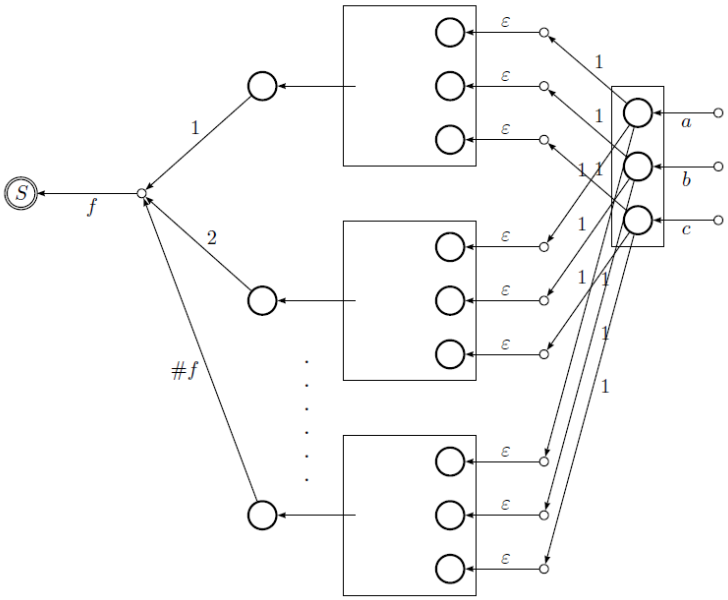
L'automate élémentaire Pour une ER $E = a$, on a $Q^E = \{S\}, \Delta^E = \{(a, S)\}$.



L'automate pour la fonction d'arité :

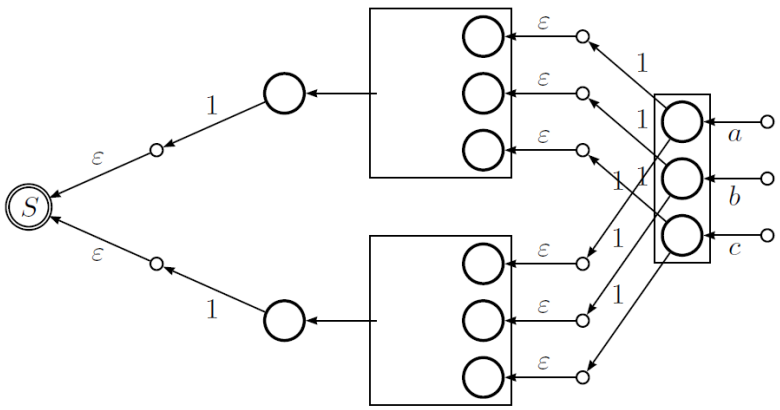
Soit $E = f(E_1, \ldots, E_{\hat{f}})$ une ER. ($E_1, \ldots, E_{\hat{f}}$ sont aussi des ER). alors : L'automate de E est construit á partir des automates élémentaires de $E_i, i \in 1, \ldots, \hat{f}$ comme suit :

$$\begin{aligned} Q^E &= \cup\{Q^{Ei}\} \cup \{q_f^E\} \cup \{q_a^E\}, \\ \Delta^E &= \{\Delta^{Ei}\} \backslash \{a \longrightarrow q_a^{Ei}, a \in \Sigma_0\} \\ &\quad \cup \{f(q_f^{E1}, q_f^{E2}, ..., q_f^{En}) \longrightarrow q_f^E\} \\ &\quad \cup \{a \longrightarrow q_a^E \backslash a \in \Sigma_0\} \\ &\quad \cup \{\epsilon(q_a^{Ei}) \longrightarrow q_a^E \backslash a \in \Sigma_0, i : 1...n\} \end{aligned}$$



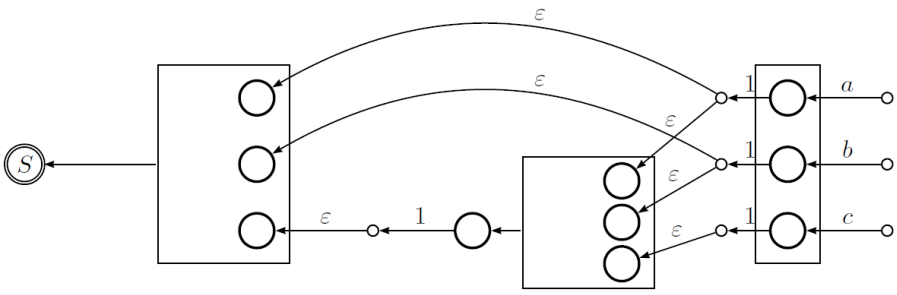
Automate de la somme Pour une ER $E = F + G$, l'automate résultant est construit á partir des deux automates élémentaires de F et de G comme suit :

$$
\begin{aligned}
Q^E &= \cup\{Q^F\} \cup \{Q^G\} \cup \{q_f^E\} \cup \{q_a^E\}, \\
\Delta^E &= \{\Delta^F\} \backslash \{a \longrightarrow q_a^E, a \in \Sigma_0\} \cup \{\Delta^G\} \backslash \{a \longrightarrow q_a^E, a \in \Sigma_0\} \\
&\quad \cup \{\epsilon(q_a^F) \backslash a \in \Sigma_0\} \\
&\quad \cup \{\epsilon(q_a^G) \backslash a \in \Sigma_0\} \\
&\quad \cup \{\epsilon(q_f^F) \longrightarrow (q_f^E)\} \\
&\quad \cup \{\epsilon(q_f^G) \longrightarrow (q_f^E)\} \\
&\quad \cup \{a \longrightarrow (q_a^E) \backslash a \in \Sigma_0\}
\end{aligned}
$$



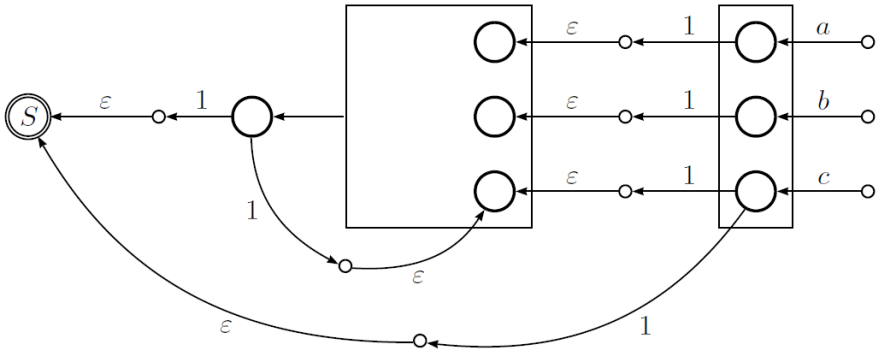
Automate de la concaténation Pour une ER $E = F._c G$,l'automate résultant est construit á partir des deux automates élémentaires de F et de G comme suit :

$$
\begin{aligned}
Q^E &= \{Q^F\} \cup \{Q^G\} \cup \{q_f^E\} \cup \{q_a^E \backslash a \in \Sigma_0\}, \\
\Delta^E &= \{\Delta^F\} \backslash \{a \longrightarrow q_a^E, a \in \Sigma_0\} \cup \{\Delta^G\} \backslash \{a \longrightarrow q_a^E, a \in \Sigma_0\} \\
&\quad \cup \{a \longrightarrow (q_a^E) \backslash a \in \Sigma_0\} \\
&\quad \cup \{\epsilon(q_a^F) \backslash a \in \Sigma_0\} \\
&\quad \cup \{\epsilon(q_a^G) \backslash a \in \Sigma_0\} \\
&\quad \cup \{\epsilon(q_f^G) \longrightarrow (q_f^c)\}
\end{aligned}
$$



Automate de l'étoile Pour une ER $E = F^{*c}$,l'automate résultant est construit á partir de l'automate élémentaire de F comme suit :

$$\begin{aligned}
Q^E &= \{Q^F\} \cup \{q_f^E\} \cup \{q_a^E \backslash a \in \Sigma_0\}, \\
\Delta^E &= \{\Delta^F\} \backslash \{a \longrightarrow q_a^E, a \in \Sigma_0\} \cup \{\Delta^G\} \backslash \{a \longrightarrow q_a^E, a \in \Sigma_0\} \\
&\quad \cup \{a \longrightarrow (q_a^E) \backslash a \in \Sigma_0\} \\
&\quad \cup \{\epsilon(q_c^E) \longrightarrow (q_f^F)\} \\
&\quad \cup \{\epsilon(q_c^F) \longrightarrow (q_f^E)\} \\
&\quad \cup \{\epsilon(q_f^F) \longrightarrow (q_f^E)\}
\end{aligned}$$



Enfin, on a :

Soit E une ER . Alors $L(\mathcal{T}_E) = [\![E]\!]$ Une opération importante est la suppression des epsilons qui existent dans l'automate de Thompson

Définition 2.6 *Soit t un arbre, nous définissons la fonction d'élimination des* $\epsilon - transitions$ *de l'arbre t,*$\epsilon - free(t)$ *comme suit :* $\epsilon - free(a) = a/a \in \Sigma_0$

$\epsilon - free(\varepsilon t) = t$

$\epsilon - free(f(t1, t2, ..., tn)) = f(\epsilon - free(t1), \epsilon - free(t2), ..., \epsilon - free(tn))$

À partir de la définition de $\epsilon - free(t)$ nous pouvons déduire la propriété suivante:

Propriété 2.7 $\epsilon - free(t1._c t2) = \epsilon - free(t1)._c \epsilon - free(t2)$. *Cette propriété peut être aussi étendue sur les ensembles .*

Propriété 2.8 $\epsilon - free(t.c\{t1, t2, ...tk\}) = \epsilon - free(t)._c\{\epsilon - free(t1), \epsilon - free(t2), ...\epsilon - free(tk)\}$

3.2 Automate de positions

L'automate de position pour les arbres est une généralisation naturelle de celui sur le mot (aussi connu sous le nom de l'automate de Glushkov [18]). Récemment, plusieurs publications ont défini et améliorer cette construction ([20]). On présente les notions utilisé pour la construction e l'automate de positions.

Automate de position

les mêmes étapes avec l'automate de Glushkov, l'automate de positions pour les arbres est construit utilisant la notion de $First$ et $Follow$ et $last$

Définition 2.9 { *Expression linéarisée* } *Une ER E travers un alphabet* Σ *est* linéaire *si chaque symbole de* $\Sigma_{\geq}$ *figure au plus une seule fois dans E. Néanmoins, cette condition est loin d'être vérifiée par toute ER. Pour cette raison, une* linéarisation *de l'ER E (notée* $\overline{E}$*) est faite par réindexation des symboles avec arité supérieur ou égale* 1.
Soit $E = f(a + g(b)^{*c}, g(c))._c(g(f(a, g(b)^{*b})))$. *En indéxaant les symboles avec arité supérieure ou égale á* 1 *on obtient :* $\overline{E} = f_1(a + g_2(b)^{*c}, g_3(c))._c(g_4(f_5(a, g_6(b)^{*b})))$

Définition 2.10 *Soit E une ER linéaire sur un alphabet Σ. La fonction $First$ est définie comme suit :*

$$First(E) = \{\mathcal{R}(t) \mid t \in [\![E]\!]\} \tag{2.1}$$

De la même manière, on définit la fonction $Follow$:

Définition 2.11 *Soit E une ER linéaire sur un alphabet Σ. La fonction $Follow$ est définie comme suit :*

$$Follow(E, f, k) = \{g \mid \exists t = f(t_1, \ldots, t_{\hat{f}}) \in [\![E]\!], \mathcal{R}(t_k) = g\} \tag{2.2}$$

Enfin, la fonction $Last$ est définie comme suit :

Définition 2.12 *Soit E une ER linéaire sur un alphabet Σ. La fonction $Last$ est définie comme suit :*

$$Last(E) = \{a \in \Sigma_0 \mid \exists t \in [\![E]\!], a \in St(t)\} \tag{2.3}$$

Par la suite, l'automate de positions est construit comme suit :

Définition 2.13 *Soit E une ER linéaire sur un alphabet Σ. l'automate de position $\mathcal{P}_E = (Q, \Sigma, Q_f, \Delta)$ de l'ER E est défini comme suit :*

$$\begin{aligned}
Q &= \{f^k \mid f \in \Sigma_m, 1 \leq k \leq m\} \\
&\quad \cup \{\varepsilon^1\} \\
Q_f &= \{\varepsilon^1\} \\
\Delta &= \{(f^k, g, g^1, \ldots, g^n) \mid f \in \Sigma_m, k \leq m, g \in Follow(E, f, k)\} \\
&\quad \cup \{(\varepsilon^1, f^1, \ldots, f^m) \mid f \in First(E)\} \\
&\quad \cup \{(\varepsilon^1, c) \mid c \in \Sigma_0 \cap First(E)\} \\
&\quad \cup \{(f^k, c) \mid f \in \Sigma_m, c \in \Sigma_0 \cap First(E)\}
\end{aligned}$$

Cette construction nous mène au résultat suivant:
Soit E une ER. Alors $L(\mathcal{P}_E) = [\![E]\!]$.

3.3 Automate des dérivées partielles

L'automate des dérivés pour les mots á été introduit par Brzozowski [21] pour la première fois la notion des dérivés d'ER pour les mots. Ensuite, Antimirov [22] à améliorer cette construction et a proposé la notion des dérivés partielles. Dans le cas des arbres, Kuske et al. [23] ont généralisé la notion des dérivées partielles pour les ER d'arbres. Ainsi, la construction d'un automate dit *automate d'équations* est permise. Soit $\mathcal{N}$ un ensemble de n-tuples d'ER à travers un alphabet Σ pour un entier $n \geq 1$. On définit la *n-iéme concaténation* avec l'ER F comme suit :

Définition 2.14

$$\mathcal{N}._c F = \{(E_1 ._c F, \ldots, E_n ._c F) \mid (E_1, \ldots, E_n) \in \mathcal{N}\} \tag{2.4}$$

Maintenant, on introduit la notion des *dérivées partielles* comme définie dans [23].

Définition 2.15 *La dérivée partielle g^{-1} d'une ER E à travers un alphabet Σ pour*

un symbole $g \in \Sigma_>$ est l'ensemble des n-tuples défini par induction comme suit :

$$\begin{aligned}
g^{-1}(f(E_1,\ldots,E_n)) &= \begin{cases} \{(E_1,\ldots,E_n)\} \textit{ si } f = g \\ \emptyset \textit{ sinon} \end{cases} \\
g^{-1}(E+F) &= g^{-1}(E) \cup g^{-1}(F) \\
g^{-1}(E^{*c}) &= g^{-1}(E).{}_c E^{*c} \\
g^{-1}(E.{}_c F) &= \begin{cases} g^{-1}(E).{}_c F \textit{ si } c \notin [\![E]\!] \\ \\ g^{-1}(E).{}_c F \cup g^{-1}(F) \textit{ sinon} \end{cases}
\end{aligned}$$

L'ensemble $\widetilde{\mathcal{N}}$ associé à un n-tuple d'ER est $\widetilde{\mathcal{N}} = \{F \mid \exists (F_1,\ldots,F,\ldots,F_n) \in \mathcal{N}, n \in \mathbb{N}\}$ La dérivée de E (notée $\partial(E)$) pour un ensemble de symboles de $\Sigma^*_{\geq 1}$ est défini par induction comme suit :

$$\begin{aligned}
\partial_\epsilon(E) &= \{E\} \\
\partial_{wg}(E) &= \cup_{E' \in \partial_w(E)} \widetilde{g^{-1}E'}, w \in \Sigma^*_{\geq 1}, g \in \Sigma_{\geq 1}
\end{aligned}$$

L'automate d'équation pour une ER E á travers Σ est l'automate défini comme suit :

Définition 2.16 *Soit E une ER définie sur un alphabet Σ. L'automate d'équations $\mathcal{E}_E = (Q, \Sigma, Q_f, \Delta)$ est défini comme suit :*

$$\begin{aligned}
Q &= \partial_{\Sigma^*_{\geq 1}}(E) \cup \{E\} \\
Q_f &= \{E\} \\
\Delta &= \{(f, G_1,\ldots,G_m,F) \mid F \in Q, f \in \Sigma_m, (G_1,\ldots,G_m \in f^{-1}(F))\} \\
&\quad \cup \{(F,c) \mid F \in Q, c \in [\![F]\!] \cap \Sigma_0\}
\end{aligned}$$

On aura :

Soit E une ER. Alors $L(\mathcal{E}_E) = [\![E]\!]$.

4 Conclusion

Dans ce chapitre étudiée l'automate Thompson , Position et Équation pour les mots et les arbres .Le prochain chapitre portera sur une étude comparative entre les différentes constructions suivant des critères bien définis

Chapitre 3

Comparision entre les construction d'automates d'arbre à partir d'un ER

1 Introduction

Dans ce chapitre , nous présentons la comparaison entre les différents construction d'automates on fixer les critère de comparaison

Critères de comparaison

Pour bien porter l'étude comparative,nous fixons les critère suivants

- Complexité temporelle et spatiale : Comparer la complexité asymptotique des différentes constructions. la meilleur construction est celle avec un complexité pire cas modérée.
- Nature des automates: L'automate résultant est il déterministe, non déterministe, avec ou sans des $\epsilon - transitions$.
- Etapes de construction: Chercher des relations entre les étapes de construction des différentes techniques. Permet d'identifier l'équivalence élémentaires entre ces constructions
- Taille de l'automate résultant : L'occupation mémoire des automates résultant. il faut constater que cette propriété ne coïncide pas avec la complexité spatiale car dans cette dernière, l'occupation mémoire intermédiaire est prise en compte
- complexité de reconnaissance : le temps nécessaire pour q'un automate reconnaisse un arbre quelconque.

2 Complexité

On reporte les complexités des constructions pour les mots (Praden Florian [24]) et on les compare avec les complexités trouvées pour les même techniques sur les arbres ([23], [20]). Les résultats sont résumés sur le tableau suivant (3.1).

Complexité \ l'automate	complexité pour mot	complexité par arbre
Thompson	O(m)	$O(2^n)$
position	O(mn)	$O(\| E \|^2)$
Dérivé partielles	$O(mlogn + mn)$	$O(R, \| E \|^2)$

Table 3.1: Complexité des automates par les mots et les arbres

3 Nature des automates résultatS

On classe les techniques de construction selon la nature des automates résultants. On rappelle que tous les automates résultants sont ascendants.

Nature \ automate	NFTA	avec $\epsilon - transition$
Thompson	✓	✓
position	✓	×
Dérivé partielles	✓	×

Table 3.2: Nature des automates par NFTA et avec $\epsilon - transition$

4 Étape de construction

Dans cette partie , nous montons l'équivalente entre les étapes de construction élémentaires dans Thompson , position et dériver .

soit E une Expression rationnelle ,cette comparaison se fais par induction sur la structure de E la preuve ce fait sur les opération suivant:

- l'expression de feuil $E = a$

- la fonction arité $E = f(E_1, ..., E_n)$

- l'addition $E = E_1 + E_2$

- l'étoile de kleene $E = E_1^{*c}$

- la concaténation $E = E_1 ._c E_2$

4.1 Construction

les expressions rationnelles $E = a$ l'Automate Thompson , l'Automate position et automate dérivé sont équivalents pour l'expression $E = a$

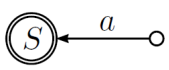
Lemme 3.1 *:*

soit $E = a$ une expression rationnelle, alors :

$T_a = P_a = \mathcal{E}_a$

Preuve

1. $T_E \Longrightarrow \{E = a\}$



2. $P_E \Longrightarrow E = a$

   . $First(a) = a$

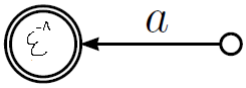
   .$Follow(E, a, 0) = 0$

   .$last(a) = a$

   .$Q = \{a\}$

   .$Q_f = \{\epsilon^1\}$

   .$\Delta = \{(a, \epsilon^1)\}$

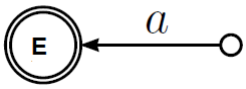


3. $\mathcal{E}_E \Longrightarrow E = a\ .\ a^{-1}(a) = \{\}$

   .$Q = \partial_{\Sigma \geq 1} = 0$

   .$Q_f = \{E\}$

   .$\Delta = \{(a, \{E\})\}$



On remarque que la construction est la même pour toutes les techniques

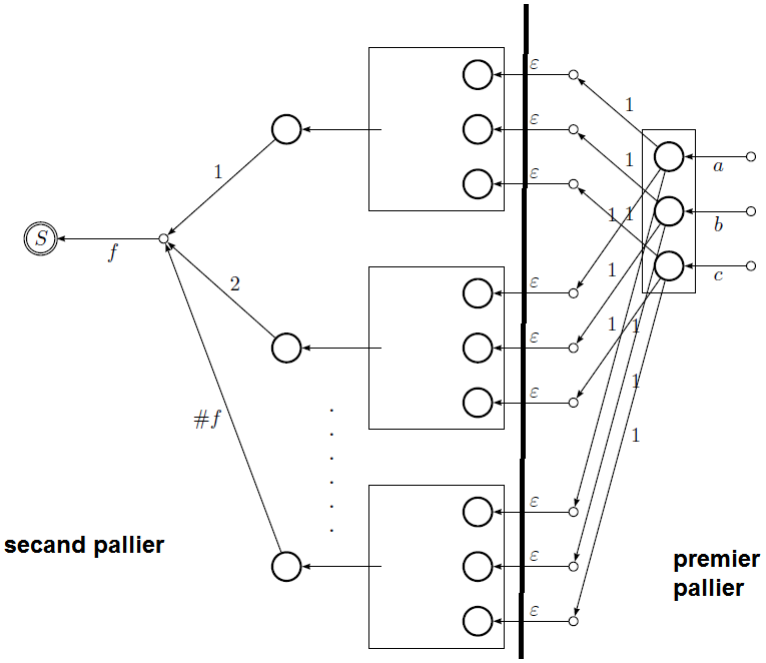
Lemme 3.2 *:*

soit $(E_1, E_2, ..., E_n)$ *Expression rationnelle,soit* $f \in \Sigma_n$,

$$T_{f(E_1,....,E_n)} \equiv P_{f(E_1,....,E_n)}$$

Preuve

c'est $T(f(E_1,....,E_n))$ est l'automate



On ajoute : $(f, S_{E1},.....S_{En}, S)$

pour le automate de position : $P(f(E_1,....,E_n))$

soit $f(E_1,.....,E_n)$ une Expression rationnelle

$First(f(E_1,.....,E_n)) = f$

$Follow(f(E_1,.....,E_n), f) = \{First(E_i), 1 \leq i \leq n\}$

l'automate $\Delta(P_{(E1)}) = \cup_{1\leq i\leq n}\Delta P(E_i) \cup (f, S_1,....., S_n) \longmapsto (1)$

$S_i = First(E_i), 1 \leq i \leq n$

$P(E_i) \equiv T(E_i), 1 \leq i \leq n$ (Hypothèse d'indication)

Remarque 3.3 *:*

On remarque que l'alphabet passe directement du premier pallier vers le second

pallier avec de $\epsilon - free$ on assure que l'automate résultant est équivalent avec l' automate de position

On peut de la même manière prouver que

Lemme 3.4 *:*

$$T(E_1 + E_2) \equiv P(E_1 + E_2)$$

$$T(E_1 ._c E_2) \equiv P(E_1 ._c E_2)$$

et

$$T(E_1^{*c}) \equiv P(E_1^{*c})$$

pour deux Expressions rationnelles E_1, E_2 en supposent que $T(E_1) \equiv p(E_1)$ et $T(E_n) \equiv p(E_n)$

Lemme 3.5 *:*

soit $E_1, ..., E_n$ expressions rationnelles. Alors : , $P(f(E_1, ..., E_n)) \equiv \mathcal{E}(f(E_1, ..., E_n))$

Preuve

- pour l'automate de position (P) il y'a:
  $First(P(f(E_1, ..., E_n))) = f$
  $Follow(f(E_1, ..., E_n)) = \cup_{1 \leq i \leq n} First E_i,$

- pour l'automate de dérivée partielle :
  $f^{-1}(f(E_1, ..., E_n)) = \{(E_1, ..., E_n)\}$
  $(E_1, ..., E_n) \in f^{-1}(E)$
  $(f, E_1, ..., E_n, E)$ on s'ajouté à Δ

$\Delta\mathcal{E}(E) = \cup_{1\leq i\leq n}\Delta\mathcal{E}(E_1) \cup (f, E_1, ..., E_n, E) \longmapsto$ (2)

donc par (1) et (2) conclure que

$P_E \equiv \mathcal{E}_E \longmapsto$ (3) On a aussi

Lemme 3.6 *:*

SoientE_1 *et* E_n*deux expressions rationnelles tel que :* $P(E_1) \equiv \mathcal{E}(E_1)$ *et* $P(E_2) \equiv \mathcal{E}(E_2)$

Preuve

on a $First(E_1 + E_2) = First(E_1) \cup First(E_2)$

et $f^{-1}(E_1 + E_2) = f^{-1}(E_1) \cup f^{-1}(E_2)$

donc par (3) on conclue que $P(E_1 + E_2) \equiv \mathcal{E}(E_1 + E_2)$

On peut de la même manière prouver que

Lemme 3.7 *:*

Soient $(E_1$ *et*$E_n)$ *deux expressions rationnelles* $P(E_1) \equiv \mathcal{E}(E_1) \wedge P(E_2) \equiv \mathcal{E}(E_2)$

preuve que $P(E_1 ._c E_2) \equiv \mathcal{E}(E_1 ._c E_2)$ *preuve que* $P(E_1^{*c}) \equiv \mathcal{E}(E_1^{*c})$

Alors, en utilisent lemme(3.1) ,lemme(3.3), lemme(3.6), lemme(3.7) ,lemme(3.9) et lemme(3.11) on a:

Théorame: 3.8 *Les automates de Thompson , Position et Dérivées son équivalents par construction*

5 Taille de l'automate résultant

Nous nous intéressons à l'occupant de chaque automate dans la mémoire Dans le

l'automate / Taille	taille par mot	taille par arbre
Thompson	$2\times \mid E \mid$	$R \mid E \mid$
position	$\parallel E \parallel^2$	$\parallel E^2 \parallel$
Dérivé partielles	$\parallel E \parallel + \parallel E \parallel^2$	$\mid E \mid \times \parallel E \parallel^2$

Table 3.3: Taille des automates par les mots et les arbres

tableau suivant, nous montrons la taille de chaque automate résultant en fonction de la taille de l'expression rationnelle de départ. Nous pouvons constater que les tailles sont pratiquement les mêmes dans les deux cas, mots et arbres à part la considération du nombre d'arité dans le second cas.

6 Complexité de reconnaissance

Étape utile après la construction,

l'automate / Complexité	complexité
Thompson	Rt
position	2^t
Dérivé partielles	2^t

Table 3.4: Complexité de reconnaissance

7 Tableau récapitulatif

l'automate / les données	complexité Temporelle	Nature	Étape de construction	Taille	complexité reconnaissons
Thompson	$O(2^n)$	FTA,$\epsilon - transition$	équivalent après free	$R \mid E \mid$	Rt
position	$O(\| E \|^2)$	FTA,	équivalent	$\| E^2 \|$	2^t
Dérivé partielles	$O(R, \mid E \mid^2)$	FTA	équivalent	$\mid E \mid \times \| E \|^2$	2^t

Table 3.5: Tableau récapitulatif résultant

On peut constater qu'a travers la construction il s'est avéré que l'automate de position et celui de dérives sont meilleurs en se referont sur la complexité de construction qui est minime. de plus, un autre inconvénient sur les automates de Thompson est la présence des $\epsilon - transition$. ce qui nécessite une tapes de suppression de ce type de transition (free).néanmoins la supériorité de l'automate de Thompson vient sur la partie reconnaissons que est en effet l'opération la plus importante et qui construite le but d'utiliser les automates.

8 Conclusion

On a effectué un tour des méthodes de construction d'automates à partir d'une expression régulière.

On a vu que l'on pouvait toujours partir de l'automate de Thompson, de le déterministe et puis le minimiser. Cependant, ce n'est pas la méthode la plus efficace. On a donc vu que l'algorithme de position permet déjà de construire un automate plus petit que celui de Thompson. Puis la construction Dériver permet d'avoir encore un automate beaucoup plus petit. En effet, cet algorithme est équivalent à une minimisation sur une expression régulière légèrement transformée

Conclusion générale

LA principale contribution de ce mémoire est la comparaison entre les techniques des automates de Thompson, Position et Dériver . Ce travail nous a permis de découvrir le meilleure technique ainsi que leur complexité Après une introduction des notions de base des langages réguliers d'arbre, ses différentes représentations et ses applications, nous avons présenté les techniques existante pour la conversion d'une expression rationnelle en un automate équivalent. Dans le troisième chapitre du mémoire nous avons exposons, après un rappel de ce nous avons étudions dans les chapitres précédent nous résultants la plus pratique technique parmi les trois techniques a partir de les comparer entres eux avec leurs complexité, natures, étapes de construction, tailles et complexités de reconnaissance afin de notre étude on se trouve que technique de Thompson est le meilleure et le plus simple pour la manipulation des automates d'arbre. et de taille et complexité minimum entre les autres techniques.

Bibliographie

[1] Silvano Zilio and Denis Lugiez. *Xml schema, tree logic and sheaves automata. In Robert Nieuwenhuis, editor, Rewriting Techniques and Applications, volume 2706 of Lecture Notes in Computer Science, pages 246–263. Springer Berlin Heidelberg*. 2003. 1

[2] Boris Chidlovskii. *Using regular tree automata as XML schemas. In Proc. IEEE Advances on Digital Libraries Conference 2000, pages 89–98*. 1

[3] Marc Tommasi. *Structures arborescentes et apprentissage automatique. PhD thesis, Universit´e Charles de Gaulle - Lille 3,*. 1

[4] J.E Doner. *Decidability of the weak second-order theory of two successors. Notices of American Math. Society,*. 1

[5] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. Available on: `http://www.grappa.univ-lille3.fr/tata`, 2008. release Novembre, 18th 2008. 5

[6] Loek Cleophas. *Tree Algorithms : Two Taxonomies and a Toolkit,*. PhD thesis, Eindhoven University of Technology. april 2008. 5, 7

[7] N.Kalyani K.Sunitha. Formal languages automata theory.mcgraw-hilltle, education (india) pvt limited,2010. 5

[8] A.A. Puntambekar. Formal languages and automata theory. technical publications ,2009. 5

[9] Joost Engelfriet. *Bottom-up and top-down tree transformations— a comparison. Mathematical systems theory,9(2) : 198¢231;*. 1975. 8

[10] URL https://fr.m.wikipedia.org/wiki/automate_fini. 9

[11] Belabbaci Ahlem. *Recharche des motifes d'arbre,Master's thesis*. PhD thesis, University, Universit´e de Laghouat, 2014. 10, 28

[12] S. Kleene.. *Representation of events in nerve nets and finite automata. Automata Studies, Ann. Math. Studies 34 :3–41, 1956. Princeton U. Press.* 13

[13] Joost Engelfriet and Erik Meineche Schmidt. *IO and OI. I. J. Comput. Syst. Sci., 15(3) :328–353,*. 1977. 16

[14] Z. Esik. *Axiomatizing the equational theory of regular tree languages. The Journal of Logic and Algebraic Programming, 79(2) :189 – 213*. 2010. 16

[15] Franz Baader and Tobias Nipkow. . *Term rewriting and all that. Cambridge University Press,*. 1998. 16

[16] Ken Thompson. *Programming techniques : Regular expression search algorithm. Commun. ACM, 11(6) :419–422,*. June 1968. 21, 28

[17] ENS Cachan CNRS Benjamin MONMEGE et Sylvain SCHMITZ LSV. *Notes de révision : Automates et langages*. URL http://creativecommons.org/licenses/by-nc-sa/2.0/fr/. Version du 24 octobre 2011. 21

[18] V.M.: Glushkov. The abstract theory of automata.russian mathematical surveys. 1961. 22, 33

[19] Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. ire trans. electronic computers, 9(1) :39–47. 1960. 22

[20] Nadia Ouali Sebti Ludovic Mignot and Djelloul Ziadi. *K-Position, Follow, Equation and K-C-Continuation Tree Automata Constructions,LITIS, Universit´e de Rouen, 76801 Saint-´ Etienne du Rouvray Cedex, France*. 33, 40

[21] J.A. Brzozowski. *Derivatives of regular expressions. Journal of the Association for Computing Machinery, 11(4) :481–494*. 1964. 35

[22] V.: Antimirov. *Partial derivatives of regular expressions and finite automaton constructions. Theoretical Computer Science, 155:291–319, 1996*. 35

[23] Dietrich Kuske and Ingmar Meinecke. *Construction of tree automata from regular expressions,RAIRO - Theoretical Informatics and Applications (2011)*. 35, 40

[24] Praden Florian. *Rapport sur les expressions rationnelles (régulières) et les automates*. 4 juin 2007. 40