Chapitre 1

Introduction

Un langage est un ensemble de mots, et un mot une suite finie de lettres. L'étude des langages est donc une étude d'objets ayant une structure linéaire et discrète (les symboles peuvent être vus comme des signaux envoyés en des instants discrets). Le mot « formel » désigne ce qui peut être traité par une machine ou un modèle mathématique.

1.1 Quelques repères historiques

1.1.1 XVIIème siècle : les premières machines

C'est un domaine relativement ancien :

- la première machine à calculer est attribuée à Schickart et a été conçue en 1623. C'était une horloge à calcul qui pouvait effectuer les quatre opérations arithmétiques. L'unique exemplaire de cette machine a brûlé en 1624 ;
- en 1642, Pascal construit sa pascaline, capable d'effectuer des additions et des soustractionws. Cette machine a été commercialisée (quelques dizaines d'exemplaires) ;
- la machine de Leibniz, conçue en 1673 et construite en 1694 pouvait réaliser les quatre opérations arithmétiques. Cette machine était plus complexe, mais ne fonctionnait pas toujours très correctement (comme la pascaline, d'ailleurs).

1.1.2 XIXème siècle : vers l'ordinateur

Au cours des XVIIIème et XIXème siècle, les machines se complexifient. Au cours du XIXème, on voit apparaître les ancêtres de l'ordinateur :

- le mathématiciens Babbage conçoit sa machine analytique entre 1834 et 1836. Cette machine fonctionne au moyen de picots sur un cylindre et peut être vue comme l'ancêtre des cartes perforées. Les premiers

programmes informatiques ont été conçus sur cette machine par Ada A. Lovelace (fille de Lord Byron) ;
– enfin les machines se perfectionnent jusqu'à l'invention de l'ordinateur due à Von Newmann (1946).

1.1.3 XXème siècle : les modèles

L'étude des modèles a été beaucoup plus tardive :
– machines de Türing (1936) ;
– Chomsky, avec la linguistique scientifique (besoin de décrire les langues naturelles), définit la notion de grammaire générale (années 1960) ;
– Kleene définit la notion d'automate (années 1960).

1.2 Problèmes classiques et applications

Les but des études des langages formels est de proposer des modèles finis (machines, automates...) pour représenter des ensembles *a priori* infinis.

1. Décider : `mot` $\mapsto$`vrai / faux`.
   – reconnaissance de motifs (traitement de texte) ;
   – fouille de données ;
   – bio-informatique, génomique (ressemblance entre deux segments d'ADN).
2. Calculer : `mot` $\mapsto$`mot'`.
   – production de code, compilation (tramsformer un programme en Caml en langage machine) ;
   – compression /décompressions de données (`zip, unzip`).
3. Organiser / contrôler :
   – vérification automatique ;
   – supervision d'ateliers.

1.3 Bibliographie

[Sak] Jacques Sakarovitch, *Éléments de théorie des automates*, 2003, Vuibert.
[Car] Olivier Carton, *Langages formels, calculabilité et complexité*, 2008, Vuibert.
[Aut] Jean-Michel Autebert, *Théorie des langages et des automates*, 1994, Masson.
[HMU] John E. Hopcroft, Rajeev Motwani, et Jeffrey D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 2003.

Chapitre 2

Mots et langages

2.1 Alphabets et mots

2.1.1 Définitions

Définition 1. *On appelle* alphabet *tout ensemble fini non vide. Les éléments d'un alphabet sont appelés les* lettres.

Définition 2. *Soit Σ un alphabet. Un* mot *sur Σ est une suite finie de lettres de Σ. Si $u = (u_1, \ldots, u_n)$ est un mot sur Σ, alors n est la* longueur *du mot, qui est notée $|u|$. Le mot de longueur 0 (que l'on appelle le* mot vide *est noté ε et l'ensemble de tous les mots sur Σ est noté Σ^*.*

Par exemple, si $\Sigma = \{a, b\}$, il y a un mot de longueur 0, ε, deux mots de longueur 1, (a) et (b) et plus généralement, 2^n mots de longueur n.

Exemple 1. *L'alphabet utilisé par un ordinateur est $\{0, 1\}$. Lorsqu'on étudie les langages de programmation, on peut ausi considérer comme alphabet d'étude l'ensemble des mots-clés (*`begin, end, for` *...).*
En biologie, l'information génétique est codée dans un chromosome sous forme des quatre bases A,C,G,T. On peut aussi considérer que les acides aminée comme alphabet.

Considérons deux mots, $u = u_1 \ldots u_n$ et $v = v_1 \ldots v_p$. Comme les mots sont des suites finies, on a $u = v$ si et seulement si $n = p$ et $\forall i \in \{1, \ldots, n\}$ $u_i = v_i$.

Définition 3. *Soient Σ un alphabet et $u = u_1 \ldots u_n$ et $v = v_1 \ldots v_p$ deux mots sur Σ. La* concaténation *de u et v est $u.v = u_1 \ldots u_n v_1 \ldots v_p$.*

Les lettres se confondent avec les mots de longueur 1, ce qui permet de noter un mot $u = u_1 \ldots u_n$.

Proposition 1. *L'ensemble Σ^* muni de la concaténation est un monoïde libre.*

Démonstration. Libre, c'est la propriété de décomposition unique que l'on vient d'énoncer. L'élément neutre est le mot vide. Rappel : un monoïde $(A, *)$ est un ensemble A muni d'une loi interne $*$ associative et qui a un unique élément neutre. □

Connaissez-vous des monoïdes non libres ? Il suffit de considérer des groupes. L'existence d'inverses empêche en général la propriété de décomposition unique.

2.1.2 Structure des mots

Définition 4 (facteurs)**.** *Soit $u \in \Sigma^*$ un mot. On dit que v est un* facteur *de u s'il existe des mots $x, y \in \Sigma^*$ tels que $u = xvy$. Si $x = \varepsilon$, alors on dit que v est un* facteur gauche *ou* préfixe*. Si $y = \varepsilon$, alors on dit que v est un* facteur droit *ou* suffixe*.*

En général, une *sous-suite* désigne une sous-suite d'un mot : les lettres de la sous-suite ne sont pas nécessairement adjacentes.

Définition 5 (puissance d'un mot)**.** *Pour tout mot $u \in \Sigma^*$, on définit récursivement la puissance de u :*

- $u^0 = \varepsilon$ *et*
- $\forall n \in \mathbb{N}$, $u^{n+1} = uu^n = u^n u$.

Définition 6 (dépendance multiplicative)**.** *Deux mots sont multiplicativement dépendants s'ils sont puissance d'un même troisième mot : u et v sont multiplicativement dépendants s'il existe un mot w et des entiers i et j tels que $u = w^i$ et $v = w^j$.*

Voici un premier résultat sur les mots :

Proposition 2. *Deux mots commutent si et seulement si ils sont multiplicativement dépendants.*

Démonstration. Si $u = w^i$ et $v = w^j$, alors u et v commutent : $uv = w^{i+j} = vu$.

Pour l'autre sens, procédons par récurrence sur la longueur maximale du mot uv. Si $|uv| = 0$ alors $u = v = \varepsilon = u^0$. Soit n un entier strictement positif et supposons le résultat acquis pour les mots dont la concaténation a une longueur strictement inférieure à n. Considérons deux mots u et v tels que $|uv| = n$. On a les cas suivants :

- si $u = \varepsilon$ (ou de façon similaire $v = \varepsilon$), $u = v^0$ et $v = v^1$.
- si $|u| = |v|$, comme $vu = uv$, $u{=}v$ puisque l'écriture des mots est unique. Donc $u = v^1$ et $v = v^1$.
- Sinon, on peut supposer que $|u| < |v|$ sans perte de généralité. L'égalité $uv = vu$ impose que u est préfixe de v et v peut s'écrire sous la forme $v = ux$. Mais alors, $uux = uv = vu = uxu$, et par simplification,

$ux = xu$. Or x est strictement plus court que v, donc $|ux| < n$ et l'hypothèse de récurrence peut s'appliquer : il existe un mot w et des entiers i et j tels que $u = w^i$ et $x = w^j$ et par suite, $v = w^{i+j}$.

□

2.1.3 Équations en mots

Le mot $\xi[u_1/x_1, \ldots, u_n/x_n]$ est le mot obtenu à partir de ζ en remplaçant chaque lettre x_i par le mot u_i.

Définition 7 (Équation en mots)**.** *Soit Σ un alphabet (les constantes) et $V = \{x_1, \ldots, x_n\}$ un ensemble fini disjoint de Σ (les variables). Une* équation en mots *sur Σ à inconnues dans V est une paire $\{\xi, \eta\}$ de mots sur $\Sigma \cup V$, que l'on pourra noter abusivement $\xi = \eta$.*

Une solution de cette équation est une suite $(u_1, \ldots, u_n)$ de mots sur Σ vérifiant $\xi[u_1/x_1, \ldots, u_n/x_n] = \eta[u_1/x_1, \ldots, u_n/x_n]$.

Σ est l'alphabet des constantes et V celui des variables.

Exemple 2. *L'équation $xy = yx$ a pour solution l'ensemble des mots multiplicativement indépendants.*

L'équation $ax = yb$ a pour solution $\{wb, aw\}, w \in \Sigma^*$.

L'équation $ax = xb$ n'a pas de solution.

Théorème 1 (Makanin, 1977)**.** *On peut trouver une solution ou décider de l'absence de solution dans une équation en mots.*

Admis, preuve très longue (une dizaine de pages).

2.1.4 Quelques exemples en combinatoire des mots

Les mots sans carré (Thue) : on dit qu'un mot contient un carré si il possède deux facteurs identiques successifs. Par exemple, le mot 011011001 contient les carrés 11 et 00, ainsi que les carrés 011011 et 110110. Il est aisé de constater que les seuls mots en binaire n'ayant pas de carré sont $0, 1, 01, 10, 010, 101$. Des mots infinis sans carré peuvent être obtenus en considérant les points fixes de systèmes de substitution. Par exemple, Thue en 1906 donne des premiers mots sans carré sur un alphabet de 4 lettres en considérant la substitution $a/abacb$, $b/abdcb$, $c/abcdb$ et $d/abcbd$.

Régularités inévitables : pour un mot infini $x = x_1x_2...$, il existe pour tout $k \geq 1$, des entiers $n, m \geq 1$ tels que $x_n = x_{n+m} = x_{n+km}$ (progressions arithmétiques de longueur arbitraire, théorème de van Der Waerden 1927).

Mots sturmiens : un mot infini binaire est *sturmien* si pour tout $n \geq 0$, le nombre de ses facteurs de longueur n est $n+1$. Par exemple, le mot de Fibonacci *abaababaabaab*... qui est l'unique point fixe du système de substitution $a/ab, b/a$ est sturmien.

Le problème de correspondance de Post (PCP) : Soit une collection de dominos $P = \{[t_1/b_1], ..., [t_k/b_k]\}$, où les t_i et b_i sont des mots dans un alphabet Σ. Une correspondance est une suite $i_1, ..., i_l$, où $t_{i1}...t_{il} = b_{i1}...b_{il}$. Le problème est de déterminer si P possède une correspondance.

Exemple 1 : $\{[b/ca], [a/ab], [ca/a], [abc/c]\}$ a la correspondance $[a/ab][b/ca][ca/a][a/ab][abc/c]$.

Exemple 2 : $\{[abc/ab], [ca/a], [acc, ba]\}$ n'a pas de correspondance.

Le nombre n de dominos constitue la taille du problème PCP(n) considéré. La longueur (en nombre de dominos) du PCP est la longueur de la correspondance la plus courte que l'on peut construire avec le jeu de dominos donné. Par exemple, avec le jeu PCP(3) $\{[100/1], [0, 100], [1, 00]\}$, on obtient la longueur 7 en considérant la séquence de dominos suivante : 1311322 (1001100100100). Un autre exemple de taille 3 ressemblant est $\{[100/1], [0, 100], [1, 0]\}$. Sa longueur augmente pourtant considérablement et est de 75 avec 2 plus petites correspondances. L'exemple suivant de taille 4 $\{[000/0], [0/111], [11, 0], [10, 100]\}$ est de longueur 204 et n'admet qu'une plus petite correspondance.

Ce problème a été présenté par Emil L. Post en 1946 comme un problème typique indécidable, même pour des tailles données a priori. Le PCP de taille 2 est décidable (l'algorithme et sa preuve restent complexes). Le PCP de taille supérieure à 7 est indécidable. La décidabilité de PCP de taille entre 3 et 6 reste un problème ouvert. Le PCP est souvent utilisé pour prouver l'indécidabilité de nombreux problèmes par une technique de réduction. La réduction consiste à montrer que si le problème considéré était décidable, alors il en serait de même pour un PCP déjà prouvé indécidable.

De façon plus poétique, je rappelle l'holorime de Louise de Vilmorin pouvant être vu comme une instance de PCP.

```
Etonnamment monotone et lasse
Est ton âme en mon automne, hélas!
```

L'indécidabilité de PCP de taille supérieure à 7 est prouvée en se ramenant à un calcul par machine de Turing.

2.2 Langages formels

2.2.1 Définitions

Définition 8. *Soit Σ un alphabet. Un* langage formel *(ou simplement langage) est un sous-ensemble de Σ^*.*

Outre les opérations classiques sur les ensembles, on définit les opérations suivantes sur les langages.

Définition 9. – *La* concaténation *de deux langages L_1 et L_2 est le langage*

$$L_1.L_2 = \{uv \mid u \in L_1,\ v \in L_2\}.$$

– *Les* puissances *d'un langage L sont :*

$$L^0 = \{\varepsilon\} \qquad et \qquad \forall n \in \mathbb{N},\ L^{n+1} = LL^n = L^nL.$$

– *L'*étoile *(ou opération de Kleene) de L est :*

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

On voit facilement que L^* est le plus petit langage contenant L et le mot vide et qui soit stable par concaténation.

On note aussi $L_1 \cup L_2 = L_1 + L_2$.

2.2.2 Équations en langages - lemme d'Arden

On s'intéresse tout particulièrement au lemme de Arden et à sa généralisation, qui vont être très utiles par la suite.

Lemme 1 (Arden)**.** *Soient A et B deux langages. L'équation $X = AX + B$ a une plus petite solution, qui est $X = A^*B$. Cette solution est unique si ε n'appartient pas au langage A.*

Démonstration. **validité** : A^*B est bien solution : $A(A^*B) + B = A^*B$.

minimalité : Soit L une solution de l'équation. Alors $L = AL + B$. En remplaçant L par $AL + B$ à droite, on a aussi $L = A(AL + B) + B = A^2L + AB + B$. En continuant ainsi, on montre (par récurrence) que $L = A^{n+1}L + \sum_{i=0}^{n} A^iB$. Ainsi, pour tout $n \geq 0$, $A^nB \subseteq L$, et donc $A^*B \subseteq L$.

unicité : si ε n'est pas élément de A, alors soit u un mot d'un langage solution L de l'équation. Si $|u| = 0$, alors le mot vide appartient à B. Sinon, si $|u| = n$, comme on a $L = A^{n+1}L + \sum_{i=0}^{n} A^iB$, u est nécessairement dans $\sum_{i=0}^{n} A^iB$. donc $L = A^*B$. □

Lemme 2 (Arden généralisé)**.** *Le système d'équations $X_i = \sum_{j=1}^{n} A_{i,j}X_j + B_i$, $i \in \{1, \ldots, n\}$ a une solution. Celle-ci est unique si $\varepsilon \notin A_{i,j}$.*

Démonstration. On diminue d'une inconnue à chaque fois : On résout l'équation $X_n = \cdots$ en fonction des autres paramètres, et on remplace X_n par cette solution. et on diminue ainsi le nombre d'équations. □

2.2.3 Classes de langages

Différentes manières de décrire des langages :
- description verbale : L est l'ensemble des mots qui sont de longueur paire
- description ensembliste : $L = \{u \mid |u| \in 2\mathbb{N}\}$
- description par une expression : $L = ((a+b)^2)^*$
- description par une machine (automate, machine de Türing)
- description génerative (grammaires)

Langage	automate	grammaire	type
rationnel	fini	régulière	3
algébrique (hors contexte)	à pile	algébrique	2
contextuel	MT linéairement bornée	contextuelle	1
récursivement énumérable	M Türing		0

Chapitre 3

Automates finis et langages rationnels

3.1 Automates finis

3.1.1 Définitions et représentations

Définition 10. *Un* automate fini non-déterministe *est un quintuplet* $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ *où :*

- Q *est un ensemble fini d'*états *;*
- Σ *est un alphabet ;*
- $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ *est la* fonction de transition *(fonction possiblement partielle) ;*
- $q_0 \in Q$ *est l'*état initial *;*
- $F \subseteq Q$ *est l'ensemble des* états terminaux.

Représentation par table de transition

On précise l'état initial, les états terminaux et on représente simplement la fonction de transition par un tableau :

	$\dots$	a	$\dots$
$\vdots$		$\vert$	
q	$--$	$\delta(q,a)$	
$\vdots$			

Représentation graphique

On peut aussi définir l'automate $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ comme un graphe orienté étiqueté avec des états distingués. Les sommets de ce graphe sont les états de l'automate, les arêtes sont étiquetées par des lettres de Σ : l'ensemble E des arêtes étiquetées est $\{(p, a, q) \mid p \in Q, a \in \Sigma, q \in \delta(p, a)\}$. Les états

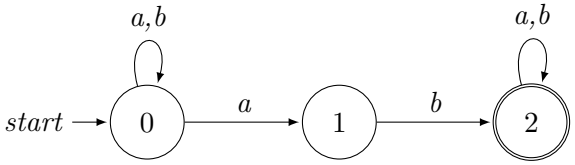
distingués sont q_0 et F. L'état initial est repéré par une flèche entrante et un état terminal par une flèche sortante (ou est doublement cerclé).

Définition 11.
- *Un* chemin *est un chemin fini dans le graphe ;*
- *un* chemin acceptant *est un chemin dont l'origine est dans I et l'(autre) extrémité est dans F ;*
- *l'*étiquette *d'un chemin est la concaténation des étiquettes des transitions qui forment le chemin ;*
- *un mot est* accepté *ou* reconnu *par l'automate $\mathcal{A}$ si c'est l'étiquette d'un chemin acceptant dans $\mathcal{A}$.*

Plus formellement, et pour introduire quelques notations, un mot $u = u_1 \dots u_n$ est accepté par $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ s'il existe une suite d'états $q_1, \dots, q_n \in Q$ tels que $\forall i \in \{1, \dots, n\}$,

$$q_i \in \delta(q_{i-1}, u_i) \qquad \text{et} \qquad q_n \in F.$$

Exemple 3. *Le mot* abaab *est reconnu par l'automate :*



Le *langage reconnu* par un automate $\mathcal{A}$ est l'ensemble des mots reconnus par cet automate. Il est noté $L(\mathcal{A})$.

Un langage est reconnaissable s'il existe un automate fini le reconnaissant. L'ensemble des langages reconnaissables sur un alphabet Σ est note $Rec(\Sigma)$.

Exemple 4.
- *Le langage vide est reconnaissable ;*
- *Pour tout alphabet Σ, Σ^* est reconnaissable ;*
- *Tout ensemble fini de mots est reconnaissable.*

Fonction de transition étendue

On peut prolonger la fonction $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ en une fonction vérifiant les propriétés suivantes :
$\delta^* : Q \times \Sigma^* \to \mathcal{P}(Q)$
- $\forall q \in Q, \delta^*(q, \varepsilon) = \{q\}$
- $\forall u \in \Sigma^*, \forall a \in \Sigma, \forall q \in Q, \delta^*(q, ua) = \bigcup_{p \in \delta^*(q,u)} \delta(p, a)$

Exemple 5. *Sur l'automate de l'exemple 3, $\delta^*(0, ab) = \{0, 2\}$*

Un mot u est alors reconnu par l'automate $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ si

$$\exists i \in I, \delta^*(i, u) \cap F \neq \emptyset$$

.

Deux types d'automates remarquables

- automate déterministe
  $\mathcal{A} = (Q, \Sigma, \delta, i, F)$, où $i \in Q$ et $\delta : Q \times \Sigma \to Q$ est une fonction partielle.
- automate déterministe complet
  $\delta : Q \times \Sigma \to Q$ est entièrement définie.
  Pour transformer un automate déterministe en automate déterministe complet il suffit de créer un état $PUITS$ et de poser $\delta(q, a) = PUITS$ pour les $\delta(q, a)$ non définis.

Définition 12.
- *Un état est dit* accessible *s'il existe un chemin d'un état initial vers cet état.*
- *Un état est dit* co-accessible *s'il existe un chemin de cet état vers un état terminal.*
- *Un automate dont tous les états sont accessibles est dit* accessible.
- *Un automate dont tous les états sont co-accessibles est dit* co-accessible.
- *Un automate accessible et co-accessible est dit* émondé.

3.1.2 Déterminisation d'automates

Proposition 3. *Il existe un algorithme qui permet de transformer un automate fini en un automate fini déterministe (complet et accessible) qui reconnaît exactement le même langage.*

Démonstration. On construit l'automate des parties.
Soit $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un automate fini qui reconnaît L. On construit $\mathcal{A}_{det} = (\mathcal{P}(Q), \Sigma, \delta_{det}, q_0, F_{det})$ tel que :
- $\forall P \in \mathcal{P}(Q), \forall a \in \Sigma, \delta_{det}(P, a) = \bigcup_{q \in P} \delta(q, a)$
- $F_{det} = P \in \mathcal{P}, P \cap F \neq \emptyset$

Il est clair que l'automate $\mathcal{A}_{det}$ ainsi construit est déterministe. Il reste donc à montrer que $L(\mathcal{A}) = L(\mathcal{A}_{det})$.
- $L(\mathcal{A}) \subseteq L(\mathcal{A}_{det})$: Soit $u \in L(\mathcal{A})$, $u = u_1 u_2 \dots u_n$ tel que $q_0 \xrightarrow{u_1} q_1 \xrightarrow{u_2} \dots \xrightarrow{u_n} q_n$, avec $q_n \in F$. Dans $\mathcal{A}_{det}$ on a $Q_0 = \{q_0\} \xrightarrow{u_1} Q_1 \xrightarrow{u_2} \dots \xrightarrow{u_n} Q_n$. Il suffit donc de montrer que $\forall i \in \{0, \dots, n\}$, $q_i \in Q_i$ et que $Q_n \in F_{det}$. La première proposition se prouve par récurrence. C'est vrai pour $i = 0$. Si on le suppose vrai pour $i - 1$, $q_{i-1} \in Q_{i-1}$. On a $q_i \in \delta(q_{i-1}, u_i) \subseteq Q_i = \bigcup_{q \in Q_{i-1}} \delta(q, u_i)$. Pour la deuxième proposition, on a $q_n \in Q_n$ donc $Q_n \cap F \neq \emptyset$ et par définition $Q_n \in F_{det}$. Donc $u \in L(\mathcal{A}_{det})$
- $L(\mathcal{A}_{det}) \subseteq L(\mathcal{A})$: Soit $u \in L(\mathcal{A}_{det}), Q_0 = \{q_0\} \xrightarrow{u_1} Q_1 \xrightarrow{u_2} \dots \xrightarrow{u_n} Q_n$ et $Q_n \in F_{det}$. Montrons par récurrence que $\forall i \in \{0, \dots, n\}, \forall q \in Q_i, \exists u = u_1 \dots u_i$ partant de q_0 et atteignant q. C'est vrai pour $i = 0$. Supposons que c'est vrai pour $i - 1$ alors, $\forall q \in Q_{i-1} \exists u_1 \dots u_{i-1}$ partant de q_0 et atteignant q. $Q_i = \delta_{det}(Q_{i-1}, u_i)$, donc $\forall p \in Q_i, \exists q \in Q_{i-1}$ tel que

$p \in \delta(q, u_i)$. Donc $q_0 \overset{u_1 \ldots u_{i-1}}{\longrightarrow} q \overset{u_i}{\rightarrow} p \in Q_i$. En particulier c'est vrai pour $i = n$ et pour $q_n \in Q_n \cap F$

Comme $L(\mathcal{A}) \subseteq L(\mathcal{A}_{det})$ et $L(\mathcal{A}_{det}) \subseteq L(\mathcal{A})$ alors on a bien $L(\mathcal{A}_{det}) = L(\mathcal{A})$ □

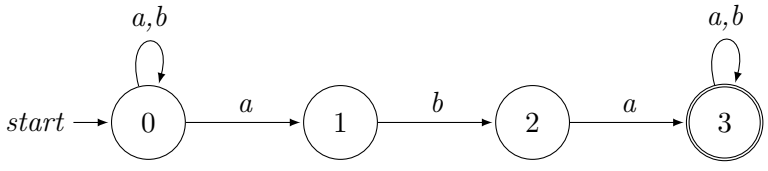
Algorithme

On en déduit un algorithme qui ne garde que les états accessibles.

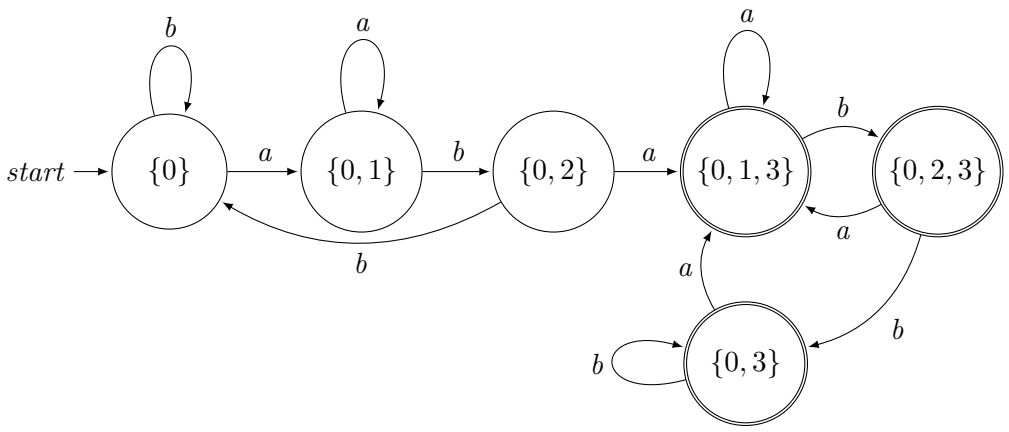
Algorithme 1 : Déterminisation d'un automate

```
Q_0 = {q_0};
Q_temp = {Q_0};
F_det = ∅;
Q_det = ∅;
δ_det = ∅;
while Q_temp ≠ ∅ do
    Soit E ∈ Q_temp;
    foreach a ∈ Σ do
        E' = ∪_{q∈E} δ(q, a);
        δ_det ← δ_det ∪ {(E, a, E')};
        if E' ∉ Q_temp ∪ Q_det then
            Q_temp ← Q_temp ∪ {E'};
            if E' ∩ F ≠ ∅ then
                F_det ← F_det ∪ E'
    Q_temp ← Q_temp \ {E};
    Q_det ← Q_det ∪ {E};
```

Exemple 6. *Soit un automate $\mathcal{A}$ représenté par :*



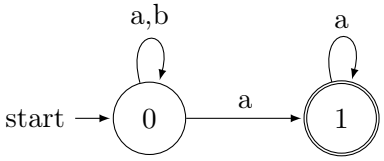
En appliquant l'algorithme à l'automate $\mathcal{A}$ on obtient l'automate déterministe $\mathcal{A}_{det}$, reconnaissant le même langage que $\mathcal{A}$, représenté par :



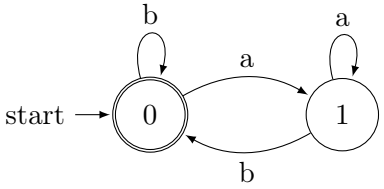
Soit un automate à $|Q|$ états. Son déterminisé a au plus $2^{|Q|}$ états. Il existe des automates pour lesquels on ne peut pas trouver d'automates déterministes avec un plus petit nombre d'états.

Exemple 7. *Le langage $\Sigma^* a \Sigma^i$: mots dont la $(i+1)^{ime}$ dernière lettre est un a. L'automate déterminisé a 2^{i+1} états et on ne peut pas faire mieux.*

Cette faculté particulière qu'ont les automates finis de pouvoir aisément se déterminiser est un atout formidable pour leur utilisation. Malheureusement, dès que l'on sort de cette catégorie, cette faculté disparaît en général. A titre d'exemple, on peut considérer les automates de Büchi qui reconnaissent des mots infinis. La seule différence avec les automates finis est l'interprétation des états terminaux qui sont vus comme des états devant être répétés infiniment. Par exemple, considérons le langage $\Sigma^* a^\omega$ formé des mots se terminant par une infinité de a. Celui n'est pas reconnaissable par un automate déterministe.



On peut noter d'ailleurs que son complémentaire (les mots ayant un nombre infini de b) est reconnaissable.



3.2 Langages rationnels

Lemme 3 (Premier lemme)**.** *Soit $L \subseteq \Sigma^*$. Si L est reconnaissable, alors il existe un entier N tel que $\forall u \in L$, $|u| \geq N$,*

$$\exists\ v,t,w \in \Sigma^* tels\ que \begin{cases} u = vtw \\ 1 \leq |t| \leq N \\ \forall k \in \mathbb{N},\ vt^k w \in L \end{cases}$$

Démonstration. Soit $L \in Rec(\Sigma)$ et $\mathcal{A} = (Q, \Sigma, \delta, I, F)$ tel que $L(\mathcal{A}) = L$. On pose $|Q| = N$. Soit $u \in L$, $|u| \geq N$, $q_o \in I$, $q_o \xrightarrow{u_1} q_1 \xrightarrow{u_2} \ldots \xrightarrow{u_n} q_n$, $n \geq N$. Il existe i,j, $i < j$ tels que $q_i = q_j$. Donc

$$q_o \xrightarrow{u_1 \ldots u_i = r} q_i \xrightarrow{u_{i+1} \ldots u_j = t} q_j = q_i \xrightarrow{u_{j+1} \ldots u_n = w} q_n.$$

□

Exemple 8. *$L_1 = \{a^p, p\ premier\}$. On suppose $L_1 \in Rec(\Sigma)$ donc $\exists N \in \mathbb{N}$ tel que en appliquant le lemme à a^p, $p \geq N$ $\exists\ l,m,n \in \mathbb{N}^3$, $a^l a^m a^n = a^p$, avec $m \geq 1$ et $l + m + n = p$ donc d'après le lemme on a $\forall k \in \mathbb{N}$, $a^{l+km+n} \in L_1$ soit $l+km+n$ premier, or pour $k = p+1$ on a $l+km+n = pm+p = p(m+1)$ qui n'est pas premier donc L_1 n'est pas reconnaisssable*

Exemple 9. *$L_2 = \{u \in \Sigma^*,\ |u|_a = |u|_b + 1\}$. Soit $u \in L_2$ tel que $|u| \geq 3$, $u \in L_2$ donc contient le facteur ab ou ba. Donc $\forall k \in \mathbb{N}$, $v(ab)^k w \in L_2$ (pour $u = vabw$). Donc on ne peut pas prouver que L_2 n'est pas reconnaissable grâce à ce premier lemme.*

Lemme 4 (Second lemme)**.** *Soit $L \subseteq \Sigma^*$. Si L est reconnaissable, alors il existe un entier N tel que $\forall u = svt \in L$, $|v| \geq N$, $\exists\ x,y,z \in \Sigma^*$ tels que $u = sxyzt$ et $1 \leq |y| \leq N$, vérifiant $\forall k \in \mathbb{N}$, $sxy^k zt \in L$*

Avec ce second lemme, on peut montrer que L_2 n'est pas reconnaissable. Pour cela on suppose que $L_2 \in Rec(\Sigma)$, alors $\exists\ N \in \mathbb{N}$ tel que pour $u = a^{N+1}b^N$ et la décomposition $x = \epsilon$, $y = a^{N+1}$, $z = b^N$. Alors $\exists\ v,t,w \in \Sigma^*$, $y = vtw$ et $|t| \geq 1$ tel que $xvwz \in L_2$, or $xvwz = a^N b^N \notin L_2$ et donc L_2 n'est pas reconnaissable.

Lemme 5 (Lemme de l'étoile par blocs)**.** *Soit $L \in Rec(\Sigma)$, alors $\exists\ N \in \mathbb{N}$ tel que $\forall u \in L$ tel que $u = u_0 u_1 \ldots u_N u_{N+1}$ avec $u_1, \ldots, u_N \in \Sigma^*$. Alors $\exists\ i,j \in \{1, \ldots, N\}$ tels que $\forall k \in \mathbb{N}$,*

$$(u_0 \ldots u_{i-1})(u_i \ldots u_{j-1})^k (u_j \ldots u_{N+1})\ avec\ N \geq |j - i| \geq 1$$

3.2.1 Stabilité des langages reconnaissables

Les énoncés qui suivent considèrent qu'il peut exister plusieurs états initiaux. Une transformation élémentaire permet de retomber sur le cas avec un seul état initial. Il suffit de rajouter un état initial q_0 et les transitions de q_0 aux états atteints directement à partir des états initiaux précédents. Attention, une simple fusion des états initiaux n'est pas correcte.

Soient $L, L_1, L_3 \in Rec(\Sigma)$

$\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ automate déterministe qui reconnaît L_1
$\mathcal{A}_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ L_2
$\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ L

– $\underline{L_1 \cup L_2 \in Rec(\Sigma)}$
$L_1 \cup L_2$ est reconnu par $\mathcal{A}_\cup = (Q_1 \cup Q_2, \Sigma, \delta_1 \cup \delta_2, \{q_{01}, q_{02}\}, F_1 \cup F_2)$

– $\underline{L_1 \cdot L_2 \in Rec(\Sigma)}$
$L_1 \cdot L_2$ est reconnu par $\mathcal{A}_\cdot = (Q_1 \cup Q_2, \Sigma, \delta_\cdot, I, F_2)$ avec

$$\begin{array}{rll} \delta_\cdot : & \text{Si } \delta_1(q,a) = p & \text{alors } p \in \delta_\cdot(q,a) \\ & \text{Si } \delta_2(q,a) = p & \text{alors } p \in \delta_\cdot(q,a) \\ & \text{Si } \delta_1(q,a) \in F_1 & \text{alors } q_{02} \in \delta_\cdot(q,a) \end{array}$$

$$\begin{array}{rll} I = & \{q_{01}\} & \text{si } \epsilon \notin L_1 \\ & \{q_{01} \cup q_{02}\} & \text{si } \epsilon \in L_1 \end{array}$$

– $\underline{L^* \in Rec(\Sigma)}$
L^* est reconnu par $\mathcal{A}_* = (Q \cup \{q_0'\}, \Sigma, \delta_*, \{q_0'\}, \{q_0'\})$ avec

$$\begin{array}{rll} \delta_* : & \text{Si } \delta(q,a) = p & \text{alors } p \in \delta_*(q,a) \\ & \text{Si } \delta(q_0,a) = p & \text{alors } p \in \delta_*(q_0',a) \\ & \text{Si } \delta(q,b) \in F & \text{alors } q_0' \in \delta_*(q,b) \\ & \text{Si } \delta(q_0,c) \in F & \text{alors } q_0' \in \delta_*(q_0',c) \end{array}$$

– $\underline{L^c \in Rec(\Sigma)}$
On suppose que $\mathcal{A}$ est complet.
L^c est reconnu par $\mathcal{A}_c = (Q, \Sigma, \delta, \{q_0\}, Q \backslash F)$

– $\underline{L_1 \cap L_2 \in Rec(\Sigma)}$
Deux preuves :

1. Par passage au complémentaire : $L_1 \cap L_2 = (L_1^c \cup L_2^c)^c$

2. $L_1 \cap L_2$ est reconnu par l'automate produit :
   $\mathcal{A}_\cap = (Q_1 \times Q_2, \Sigma, \delta_\cap, \{q_{01}, q_{02}\}, F_1 \times F_2)$ avec
   $\delta_\cap((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$

Ces résultats sont utiles pour prouver qu'un langage n'est pas reconnaissable. Notamment, si $L \cap R$ n'est pas reconnaissable et $R \in Rec(\Sigma)$ alors $L \notin Rec(\Sigma)$.

3.2.2 Langages rationnels et expressions régulières

Définition 13. *La* classe des langages rationnels sur Σ *est la plus petite classe de langages*
- *qui contient* $\{\epsilon\}$, $\emptyset$ *et* $\{a\}$ $\forall a \in \Sigma$
- *stable par* $\cup$, $\cdot$ *et* *

Définition 14. *Une* expression régulière sur Σ ($ER(\Sigma)$) *est un mot sur l'alphabet* $\Sigma \cup \{\cdots, +, ^*, \emptyset, \epsilon, (,)\}$ *tel que*
- $\forall a \in \Sigma, a \in ER(\Sigma)$
- $\epsilon, \emptyset \in ER(\Sigma)$
- *Si* α_1, $\alpha_2 \in ER(\Sigma)$ *alors* $(\alpha_1 + \alpha_2), (\alpha_1 \cdots \alpha_2), (\alpha_1^*) \in ER(\Sigma)$

Définition 15. *Un langage sur* Σ *décrit par une expression régulière est construit par l'application*

$$\begin{array}{rrcl} \lambda : & ER(\Sigma) & \to & \mathcal{P}(\Sigma^*) \\ & a & \mapsto & \{a\}\ \forall a \in \Sigma \\ & \emptyset & \mapsto & \emptyset \\ & \epsilon & \mapsto & \epsilon \\ & (\alpha_1 + \alpha_2) & \mapsto & \lambda(\alpha_1) + \lambda(\alpha_2) \\ & (\alpha_1 \cdot \alpha_2) & \mapsto & \lambda(\alpha_1) \cdot \lambda(\alpha_2) \\ & (\alpha^*) & \mapsto & (\lambda(\alpha))^* \end{array}$$

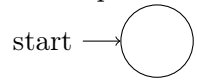
Abus d'écriture :
- Quand $\Sigma = \{a, b\}$, $a + b$ s'écrit Σ.
- $a \cdot a \cdot a$ s'écrit a^3.

Proposition 4 (Théorème de Kleene)**.** *L'ensemble des langages reconnaissables est exactement l'ensemble des langages rationnels.*

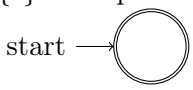
On note $Rat(\Sigma)$ l'ensemble des langages rationnels sur Σ.

Démonstration. Montrons la double inclusion.

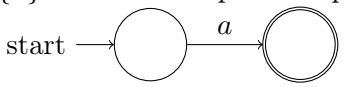
$\underline{Rat(\Sigma) \subseteq Rec(\Sigma) :}$
- $\emptyset$ est représenté par :



– $\{\epsilon\}$ est représenté par :



– $\{a\}$ $\forall a \in \Sigma$ est représenté par :



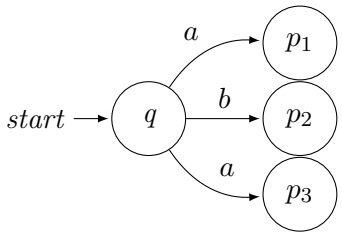
– $Rec(\Sigma)$ est stable par $\cup$, $\cdot$, * et $Rat(\Sigma)$ est la plus petite classe vérifiant ces propriétés.

$\underline{Rec(\Sigma) \subseteq Rat(\Sigma) :}$

Soit $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ un automate fini.

$\forall q \in Q$, L_q est le langage reconnu par $(Q, \Sigma, \delta, q, F)$

Exemple 10.



$L_q = a \cdot (L_{p_1} + L_{p_3}) + b \cdot L_{p_2}$

Montrons que $L_q = \sum_{p \in Q} A_{qp} \cdot L_p + X_q$

Soit $u \in L_q$. Si $u = \epsilon$, alors $q \in F$, donc $u \in X_q$ et $u \in \sum_{p \in Q} A_{qp} \cdot L_p + X_q$

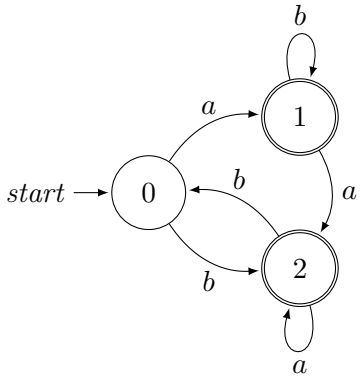
Sinon, $u = av$ et il existe p tel que $\delta(q, a) = p$ et on passe de p à un état terminal en lisant v. Donc $v \in L_p$. Donc $u \in \sum_{p \in Q} A_{qp} \cdot L_p$. La réciproque se démontre de manière analogue.

On obtient donc un système d'équations linéaires. Grâce au lemme d'Arden ($\epsilon \notin A_{qp}$), on le résoud en n'utilisant que les opérations $\cup$, $\cdot$, $+$ sur A_{qp} et X_p qui sont rationnelles.

Donc $\forall q L_q \in Rat(\Sigma)$ et $L_{q_0} \in Rat(\Sigma)$

□

Exemple 11.



$$\begin{cases} L_0 &= aL_1 + bL_2 \\ L_1 &= bL_1 + aL_2 + \epsilon \\ L_2 &= aL_2 + bL_0 + \epsilon \end{cases}$$

$$\begin{aligned} L_2 &= a^*(bL_0 + \epsilon) \\ L_1 &= bL_1 + aa^*bL_0 + aa^* + \epsilon \\ &= b^*(aa^*bL_0 + aa^* + \epsilon) \\ &= b^*aa^*bL_0 + b^*a^* \\ L_0 &= ab^*aa^*bL_0 + ab^*aa^* + ab^* + ba^*bL_0 + ba^* \\ &= (a^*baa^*b + ba^*b)^*(ab^*aa^* + ba^*) \end{aligned}$$

3.2.3 Problèmes de décisions

Les problèmes qui se posent sont les suivants :
- $L = \emptyset\,?$
- $u \in L\,?$
- $L_1 = L_2\,?$

Trouvons au cas par cas comment les résoudre.

$\boxed{L = \emptyset\,?}$

automate : Est-ce qu'il existe un état final accessible ?

expression régulière :

$$\begin{array}{rrcl} vide: & \emptyset & \mapsto & OUI \\ & \epsilon, a & \mapsto & NON \\ & \alpha + \beta & \mapsto & vide(\alpha) \cap vide(\beta) \\ & \alpha \cdot \beta & \mapsto & vide(\alpha) \\ & \alpha^* & \mapsto & NON \end{array}$$

$\boxed{u \in L\,?}$

automate : Est-ce qu'il existe un chemin d'étiquette u qui s'achève sur un état final ?

expression régulière : On ne le fait pas. En fait, le bon algorithme sera de synthétiser l'automate correspondant à l'expression avant de répondre.

$\boxed{L_1 = L_2\,?}$ Il va falloir le faire à la main, mais on peut s'aider des règles de Salomaa :

1. $(\alpha + (\beta + \gamma)) = ((\alpha + \beta) + \gamma)$
2. $(\alpha \cdot (\beta \cdot \gamma)) = ((\alpha \cdot \beta) \cdot \gamma)$
3. $((\alpha + \beta) \cdot \gamma) = ((\alpha \cdot \gamma) + (\beta \cdot \gamma))$
4. $(\alpha \cdot (\beta + \gamma)) = ((\alpha \cdot \beta) + (\alpha \cdot \gamma))$

5. $\alpha + \beta = \beta + \alpha$
6. $\alpha \cdot \epsilon = \epsilon \cdot \alpha = \alpha$
7. $\emptyset + \alpha = \alpha$
8. $\emptyset \cdot \alpha = \alpha \cdot \emptyset = \emptyset$
9. $\alpha^* = (\alpha^* \alpha + \epsilon)$
10. $\alpha^* = (\alpha + \epsilon)^*$

et des règles d'inférences :

- si $\alpha = \alpha \cdot \beta + \gamma$ et si $\epsilon \notin \lambda(\beta)$ alors $\alpha = \gamma \cdot \beta^*$
- si $\alpha = \beta$ et $\gamma = \delta$ alors $\gamma \left(\frac{\beta}{\alpha}\right) = \delta \left(\frac{\beta}{\alpha}\right)$

Exemple 12. *Montrons que* $(a+b)^* = (a^*b)^*a^*$.

$$\begin{aligned} \underbrace{(a^*b)^*a^*}_{\alpha} &= (a^*b)^*(a^*a + \epsilon) \\ &= (a^*b)^*a^*a + (a^*b)^* \\ &= (a^*b)^*a^*a + (a^*b)^*a^*b + \epsilon \\ &= \underbrace{(a^*b)^*a^*}_{\alpha}\underbrace{(a+b)}_{\beta} + \underbrace{\epsilon}_{\gamma} \end{aligned}$$

Donc, en utilisant la première règle d'inférence :

$$(a^*b)^*a^* = (a+b)^*$$

La encore, la question trouvera une réponse automatique en passant aux automates par la déterminisation et minimisation.

Test du mot vide

Soit un langage L. Calculons une fonction $\epsilon(L)$ qui rend ϵ si $\epsilon \in L$ et $\emptyset$ sinon.

- $\epsilon(\emptyset) = \emptyset$
- $\epsilon(\epsilon) = \epsilon$
- $\epsilon(a) = \emptyset$
- $\epsilon(e + e') = \epsilon(e) + \epsilon(e')$
- $\epsilon(e.e') = \epsilon(e).\epsilon(e')$
- $\epsilon(e^*) = \epsilon$

3.2.4 Résiduels (dérivées)

Définition 16. *Soient* $L \subseteq \Sigma^*$ *et* $u \in \Sigma^*$. *On appelle résiduel (à gauche) de* L *par rapport à* u *le langage* $u^{-1}L = \{v \in \Sigma^* | uv \in L\}$

Remarque : $v^{-1}(u^{-1}L) = (uv)^{-1}L$ puisque $v^{-1}(u^{-1}L) = \{w | vw \in u^{-1}L\} = \{w | uvw \in L\}$

Exemple 13. *Résiduels de* $L = \{a^n b^n | n \in \mathbb{N}\}$. *Soit* $i \in \mathbb{N}$
si $i \leq n$: $(a^i)^{-1}L = \{a^k b^{i+k} | k \in \mathbb{N}\}$ *et* $(a^n b^i)^{-1}L = b^{n-i}$

Théorème 2. *Un langage est rationnel ssi il a un nombre fini de résiduels (i.e.* $\{u^{-1}L | u \in \Sigma^*\}$ *est fini)*

Démonstration. Soit L un langage rationnel et $A = (Q, \Sigma, \delta, q_0, F)$ un automate fini déterministe complet accessible qui le reconnaît. On note $L_q = \{u \in \Sigma^* | \delta^*(q, u) \in F\}$ le langage reconnu à partir de l'état q. On pose

$$\phi : \begin{cases} \{u^{-1}L | u \in \Sigma^*\} & \to \{L_q | q \in Q\} \\ u^{-1}L & \mapsto L_{\delta^*(q_0,u)} \end{cases}$$

$w \in u^{-1}L \Leftrightarrow uw \in L \Leftrightarrow \delta^*(q_0, uw) \in F \Leftrightarrow \delta^*(\delta^*(q_0, u), w) \in F \Leftrightarrow w \in L_{\delta^*(q_0,u)}$
donc ϕ est l'identité, et, comme Q est fini, l'ensemble des résiduels de L est fini. □

Soit L un langage ayant un nombre fini de résiduels. Construisons l'automate des résiduels :
- $Q = \{u^{-1}L | u \in \Sigma^*\}$
- $q_0 = \epsilon^{-1}L = L$
- $F = \{u^{-1}L | \epsilon \in u^{-1}L\}$
- $\delta(u^{-1}L, a) = a^{-1}(u^{-1}L) = (ua)^{-1}L$

Il est déterministe, complet et accessible. Et il reconnait bien L :
$u \in L \Leftrightarrow \epsilon \in u^{-1}L \Leftrightarrow \delta^*(q_0, u) \in F$

Théorème 3 (Brzozowski 1966)**.** *Une expression régulière e n'a qu'un nombre fini de dérivées (simplifiées) distinctes* $u^{-1}(e)$

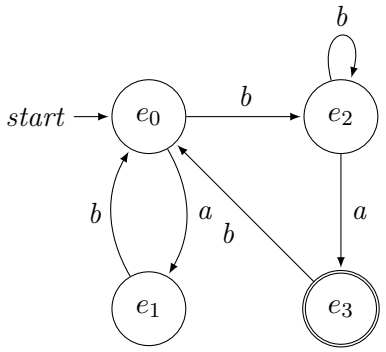
Pour calculer les dérivées, on dérive lettre par lettre. Soit $a \in \Sigma$.
- $a^{-1}(\emptyset) = \emptyset$
- $a^{-1}(\epsilon) = \emptyset$
- $a^{-1}(a) = \epsilon$
- $a^{-1}(b) = \emptyset$ si $b \neq a$
- $a^{-1}(e + e') = a^{-1}(e) + a^{-1}(e')$
- $a^{-1}(e.e') = a^{-1}(e).e' + \epsilon(e).a^{-1}(e')$
- $a^{-1}(e^*) = a^{-1}(e).e^*$

Exemple 14.
- $\begin{aligned} a^{-1}((ab+b)^*ba) &= a^{-1}((ab+b)^*)ba + \epsilon((ab+b)^*).a^{-1}(ba) \\ &= a^{-1}(ab+b).(ab+b)^*ba + \emptyset \\ &= b(ab+b)^*ba \end{aligned}$
- $\begin{aligned} b^{-1}((ab+b)^*ba) &= b^{-1}((ab+b)^*)ba + \epsilon((ab+b)^*).b^{-1}(ba) \\ &= b^{-1}(ab+b).(ab+b)^*ba + a \\ &= (ab+b)^*ba + a \end{aligned}$

Exemple 15. *Itération jusqu'à plus soif pour obtenir toutes les dérivées de* $e = e_0 = ((ab + b)^*ba)$

- $a^{-1}(e_0) = b(ab + b)^*ba = e_1$
- $b^{-1}(e_0) = (ab + b)^*ba + a = e_2$
- $a^{-1}(e_1) = \emptyset$
- $b^{-1}(e_1) = (ab + b)^*ba = e_0$
- $a^{-1}(e_2) = b(ab + b)^*ba + \epsilon = e_3$
- $b^{-1}(e_2) = (ab + b)^*ba + a = e_2$
- $a^{-1}(e_3) = \emptyset$
- $b^{-1}(e_3) = (ab + b)^*ba = e_0$

Sachant que e_3 *est le seul état valant* ϵ*, voici l'automate correspondant :*



3.2.5 Minimisation

Définition 17. *Soit* $L \in Rec(\Sigma)$ *et* $\mathcal{A}$ *un automate fini déterministe complet le reconnaissant.* $\mathcal{A}$ *est dit minimal si* $\forall \mathcal{A}'$ *fini déterministe complet reconnaissant* L, $|Q| \leq |Q'|$

Théorème 4. *L'automate des résiduels est minimal et unique à isomorphisme près.*

Définition 18 (Equivalence de Nérode)**.** *Soit* $\mathcal{A}$ *un automate fini déterministe complet.* u *sépare* q *et* q' *si* $\delta^*(q, u) \in F$ *et* $\delta^*(q', u) \notin F$ *ou inversement.* $q \sim q'$ *s'il ne sont séparés par aucun mot.*

Définition 19 (Automate des quotients)**.** *On définit l'automate des quotients comme suit (où* $\forall q \in Q$, $\overline{q}$ *désigne la classe d'équivalence de l'état* q*.)*

- $\overline{\mathcal{A}}(\overline{Q}, \Sigma, \overline{\delta}, \overline{q}_0, \overline{F})$
- $\overline{Q} = \{\overline{q}, q \in Q\}$
- $\overline{F} = \{\overline{q}, q \in F\}$
- $\overline{\delta}(\overline{q}, a) = \overline{\delta(q, a)}$

Montrons que l'automate des quotients reconnaît bien le même langage.

Soient $q_1 \sim q_2$ on a :

$\forall u \in \Sigma^*, \forall a \in \Sigma$

$\delta^*(q_1, au) \in F \Leftrightarrow \delta^*(q_2, au) \in F$

$\delta^*(\delta(q_1, a), u) \in F \Leftrightarrow \delta^*(\delta(q_2, a), u) \in F$
$\delta(q_1, a) \sim \delta(q_2, a)$

$u \in L(\mathcal{A}) \Leftrightarrow \delta^*(q_0, u) \in F \Leftrightarrow \overline{\delta^*(q_0, u)} \in \overline{F} \Leftrightarrow \overline{\delta}^*(\overline{q}_0, u) \in \overline{F} \Leftrightarrow u \in L(\overline{\mathcal{A}})$

Construction de l'automate des quotients

On note $q \sim_k q' \Leftrightarrow$ q et q' ne sont séparés par aucun mot u tel que $|u| \leq k$
Ainsi $\sim_0$ définit deux classes d'équivalence : F et $Q \setminus F$.

Proposition 5.

$$q \sim_{k+1} q' \Leftrightarrow \begin{cases} q \sim_k q' \\ \forall a \in \Sigma, \delta(q, a) \sim_k \delta(q', a) \end{cases}$$

Démonstration. ($\Rightarrow$)
- Si $q \sim_{k+1} q'$ alors en particulier $q \sim_k q'$
- Ensuite, prenons $a \in \Sigma$ et $u \in \Sigma^*$ tel que $|u| \leq k$. Alors

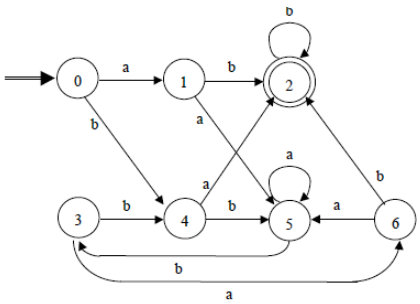
  $\delta^*(\delta(q, a), u) \in F$
  $\Leftrightarrow \delta^*(q, au) \in F$
  $\Leftrightarrow \delta^*(q', au) \in F$
  $\Leftrightarrow \delta^*(\delta(q', a), u) \in F$

  Remarquer que cela ne marche bien que parce que l'automate est déterministe.

($\Leftarrow$)
- Pour u tel que $|u| \leq k$ on a u ne sépare pas q et q'.
- Pour $|u| = k + 1$, $u = av$ avec $|v| = k$ $\delta^*(q, av) \in F \Leftrightarrow \delta^*(\delta(q, a), v) \in F \Leftrightarrow \delta^*(\delta(q', a), u) \in F \Leftrightarrow \delta^*(q', av) \in F$ on a alors $q \sim_{k+1} q'$

□

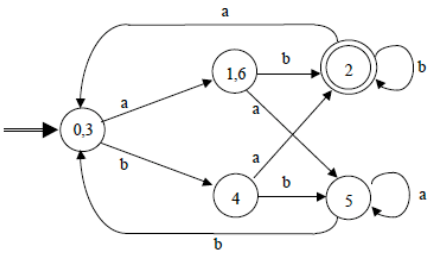
Exemple 16. *Minimisons l'automate suivant :*



Déterminons les classes d'équivalences :

$\sim_0$: $\{0,1,3,4,5,6\}\{2\}$
$\sim_1$: $\{0,3,5\}\{1,6\}\{4\}\{2\}$
$\sim_2$: $\{0,3\}\{5\}\{1,6\}\{4\}\{2\}$

On obtient donc l'automate minimisé suivant :



Noter bien qu'il n'existe pas de notion d'automate minimal dans le cas non déterministe.

3.3 Hauteur d'étoile et reconnaissance par monoïde.

3.3.1 Hauteur d'étoile

Définition 20 (hauteur d'une expression régulière)**.** *La hauteur d'étoile h d'une expression régulière est définie par récurrence par :*

- $h(\emptyset) = h(\varepsilon) = 0$;
- $\forall a \in \Sigma,\ h(a) = 0$;
- $\forall \alpha, \beta \in ER(\Sigma),\ h(\alpha + \beta) = \max(h(\alpha), h(\beta))$;
- $\forall \alpha, \beta \in ER(\Sigma),\ h(\alpha\beta) = \max(h(\alpha), h(\beta))$;
- $\forall (\alpha) \in ER(\Sigma),\ h(\alpha^*) = 1 + h(\alpha)$.

Exemple 17.
- $h((a+b)^*) = 1$;
- $h((a^*b)^*a^*) = 2$.

A un langage donné, on peut associer plusieurs hauteurs puisque plusieurs expressions régulières peuvent décrire ce même langage (comme dans l'exemple précédent). On définit donc la notion de hauteur pour les langages rationnels.

Définition 21 (hauteur d'un langage rationnel)**.** *La hauteur d'un langage rationnel L est le minimum des hauteurs des expressions régulières qui le décrivent : $h(L) = min(h(\alpha) \mid \lambda(\alpha) = L)$.*

Exemple 18.
- $h(\Sigma^*) = 1$;
- $h(L) = 0$ *si et seulement si* L *est fini.*

Existe-t-il des langages qui ont une hauteur arbitrairement grande ? (Oui, problème étudié par Schützenberg et Dejean en 1966)

Théorème 5. *Pour tout entier n, il existe un langage rationnel L tel que $h(L) = n$.*

On peut montrer cela en considérant les langages décrits par l'expression régulière suivante, définie récursivement par :
- $\alpha_0 = \epsilon$
- $\alpha_{n+1} = (a...a\alpha_n b..b)^*$ avec 2^n fois la lettre a et 2^n fois la lettre b.

Peut-on calculer la hauteur d'étoile ? (Oui : théorème d'Hashigushi 1988. Difficile !)

On peut introduire la notion de hauteur généralisée : on autorise les opérations booléennes, telles l'intersection ($\cap$) et le complémentaire (^{c}).

Définition 22 (Hauteur d'étoile généralisée)**.** *On définit récursivement la hauteur d'étoile généralisée h_g pour les expressions régulières généralisées par :*
- $\forall a \in \Sigma,\ h_g(a) = 0$;
- $\forall \alpha, \beta \in ER(\Sigma),\ h_g(\alpha+\beta) = h_g(\alpha.\beta) = h_g(\alpha \cap \beta) = \max(h_g(\alpha), h_g(\beta))$;
- $\forall \alpha \in ER(\Sigma),\ h_g(\alpha^c) = h_g(\alpha)$;
- $h_g(\alpha^*) = 1 + h_g(\alpha)$.

Il existe des langages de hauteur 0 et 1. Mais on ne connaît pas de langage de hauteur supérieur ou égale à 2 (l'existence de tels langages est un problème ouvert). Le langage $(aa)^*$ est de hauteur 1. On sait caractériser les langages de hauteur 0. Pour ce faire, on définit une nouvelle manière de reconnaître les langages rationnels : la reconnaissance par monoïde. Ils correspondent aussi à une sous-classes des automates finis, appelée les automates non compteurs. Il en existe aussi une caractérisation en logique : la logique temporelle sur les séquences finies.

3.3.2 Logique temporelle et automates finis

Considérons un alphabet Σ et l'ensemble des formules $\mathcal{F}$ que l'on peut construire par les règles suivantes :
- $0, 1 \in \mathcal{F}$
- $\forall a \in \Sigma, a \in \mathcal{F}$
- $\forall \phi \in \mathcal{F}, \neg\phi, \circ\phi \in \mathcal{F}$
- $\forall \phi, \psi \in \mathcal{F}, \phi \wedge \psi, \phi \mathcal{U} \psi \in \mathcal{F}$

On considérera les abbréviations usuelles pour $\phi \vee \psi \equiv \neg(\phi \wedge \psi)$, $\phi \Rightarrow \psi \equiv \neg\phi \vee \psi$. Ainsi que $\Diamond\phi \equiv 1\mathcal{U}\phi$ et $\Box\phi \equiv \neg\Diamond\neg\phi$.

Exemple 19. $\Box\Diamond a, \Box(a \Rightarrow \circ(\neg b \mathcal{U} a))$ *sont des formules.*

Il s'agissait là de la définition de la syntaxe. Il faut maintenant définir la sémantique (en logique on parle des "modèles"). Ici, on considère les modèles comme étant des mots finis sur l'alphabet Σ. Considérons un mot $u = a_1...a_n$ et notons $u_i = a_i..a_n$ un suffixe ($1 \leq i \leq n$). On dit qu'un mot u satisfait la formule ϕ (noté $u \models \phi$) ssi $u_1 \models \phi$. $\models$ est défini inductivement par $\forall u \in \Sigma^*, \forall i, 1 \leq i \leq |u|$:

- $u_i \models 0$ est faux
- $u_i \models 1$ est vrai
- $u_i \models a$ ssi $a_i = a$
- $u_i \models \neg\phi$ ssi $u_i \models \phi$ est faux
- $u_i \models \circ\phi$ ssi $i < |u|$ et $u_{i+1} \models \phi$
- $u_i \models \phi \wedge \phi$ ssi $u_i \models \phi$ et $u_i \models \psi$
- $u_i \models \phi \mathcal{U} \psi$ ssi $\exists j \geq i$ tel que $u_j \models \psi$ et $\forall k, i \leq k < j, u_k \models \phi$

Ainsi $\Diamond a$ désigne tous les mots qui contiennent la lettre a, $\Box\Diamond a$, les mots qui se terminent par a et $\Box(a \Rightarrow \circ(\neg b \mathcal{U} a))$ les mots tels qu'il n'y a pas de b entre deux a. Il apparaît que les langages ainsi définis par des formules de logique sont exactement les langages sans étoile généralisée et sont donc représentables aussi par des automates finis.

3.3.3 Reconnaissance par monoïde fini

Soit M un monoïde fini et soit un morphisme $\varphi : \Sigma^* \to M$. Une partie L de Σ^* est reconnue par le morphisme φ si

$$\exists P \subseteq M, \varphi^{-1}(P) = L.$$

Par abus de langage, on parle aussi de reconnaissabilité par un monoïde (L est reconnaissable par le monoïde M s'il existe un morphisme $\varphi : \Sigma^* \to M$ tel que L est reconnaissable par φ).

Théorème 6. *L est reconnaissable par un monoïde fini si et seulement si L est reconnaissable (par un automate fini).*

Démonstration. On suppose que L est reconnu par un monoïde fini. Il existe un monoïde M fini, un morphisme $\varphi : \Sigma^* \to M$ et une partie $P \subseteq M$ tels que $L = \varphi^{-1}(P)$.

Posons $\mathcal{A} = (M, \Sigma, \delta, 1_M, P)$ avec $\forall q \in M$, $\forall a \in \Sigma$, $\delta(q, a) = q \cdot \varphi(a)$.

On a :

$$(w \in L) \Leftrightarrow (w \in \varphi(P)).$$

En effet, $\delta^*(1_M, u) = \delta(\cdots(\delta(1_M, u_1), \cdots, u_n) = 1_M\varphi(u_1)\cdots\varphi(u_n) = \varphi(u)$.

Réciproquement, supposons L reconnaissable par un automate fini $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ complet et accessible. Le monoïde des transitions est Q^Q, l'ensemble des fonctions de Q dans Q. On utilise une représentation par couples $\{(q, f(q)), q \in Q\}$ ou la représentation classique des fonctions.

On définit φ par :

– $\varphi(\varepsilon) = \{(q, q) \mid q \in Q\}$;
– $\forall a \in \Sigma$, $\varphi(a) = \{(p, q) \mid q = \delta(p, a)\}$.
La loi du monoïde est la composition des fonctions :

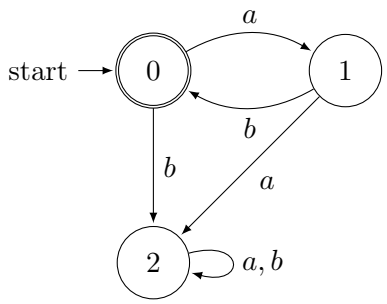
$$\begin{aligned} \varphi(w_1 \cdot w_2) &= \{(p, r) \mid p \in Q, (p, q) \in \varphi(w_1), (q, r) \in \varphi(w_2)\} \\ &= \{(p, r) \mid \delta^*(p, w_1 w_2) = r\}. \end{aligned}$$

L'ensemble P est $\{f \in Q^Q \mid f(q_0) \in F\}$.

□

Définition 23 (Monoïde syntaxique)**.** *Le monoïde syntaxique d'un langage rationnel est le monoïde des transitions associé à l'automate minimal de ce langage.*

Exemple 20. *$L = (ab)^*$. L'automate minimal associé à L est :*



On calcule le morphisme φ en considérant les fonctions $\varphi(a)$ pour chaque $a \in \Sigma$ et en fermant l'ensemble des fonctions par composition des fonctions déjà obtenues. Le nombre d'états de l'automate étant fini, le nombre de fonctions différentes obtenues est aussi fini. Lorsqu'on trouve une situation telle que $\varphi(v) = \varphi(u)$ avec $\varphi(u)$ déjà mis dans le monoïde, on dit que le monoïde possède la loi $v = u$. On désigne la fonction par le mot qui l'a construite.

φ	ϵ	a	b	aa	bb	ab	ba	aba	bab
0	*0*	*1*	*2*	*2*	*2*	*0*	*2*	*1*	*2*
1	*1*	*2*	*0*	*2*	*2*	*2*	*1*	*2*	*0*
2	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*	*2*

On a dans ce cas :
– *$M = \{\epsilon; a; b; aa; ab; ba\}$, de neutre ε avec les lois suivantes : $aaa = aa$, $bb = aa$, $aba = a$, $bab = b$;*
– *$P = \{\epsilon; ab\}$*

3.3.4 Application à la hauteur d'étoile généralisée

Définition 24 (Groupe engendré par un élément d'un monoïde fini)**.** *Soit $m \in M$. Le groupe engendré par m est $\{m^n, n \in \mathbb{N}\}$.*

Définition 25 (Période d'un groupe)**.** *Soient k et ℓ les plus petits entiers tels que $k > \ell$ et $m^\ell = m^k$.*

Le groupe associé à m est dit apériodique si $k = \ell + 1$. Sinon, $k - l$ est appelé la période du groupe.

M est apériodique si

$$\forall x \in M,\ \exists n \in \mathbb{N},\ x^{n+1} = x^n.$$

Théorème 7. *Le langage L a une hauteur d'étoile généralisé nulle si et seulement si tous les éléments de son monoïde syntaxique sont apériodiques.*

Démonstration. Admis □

Exemple 21.
1. *$(ab)^*$ est de hauteur d'étoile nulle : $\varepsilon^2 = \varepsilon$, $\varphi(aa) = \varphi(aaa)$ donc $\varphi(a)^2 = \varphi(a)^3$, $\varphi(ab) = \varphi(abab) = (\varphi(ab))^2$, $\varphi(ba) = (\varphi(ba))^2$. On peut donner l'expression généralisée de $(ab)^*$: $(ab)^* = (a\Sigma^* \cap \Sigma^* b) \setminus (\Sigma^*(aa + bb)\Sigma^*) \cup \varepsilon$.*
2. *$(aa)^*$ est de hauteur d'étoile 1 : $\varphi(\varepsilon) = \varphi(aa)$, le monoïde syntaxique est $\mathbb{Z}/2\mathbb{Z}$.*

	ε	a	aa
0	*0*	*1*	*0*
1	*1*	*0*	*1*

3.3.5 Ensembles rationnels sur un monoïde M

Nous avons vu plusieurs manières de reconnaître et caractériser les langages rationnels : les automates finis, le nombre de résiduels, les expressions rationnelles et les monoïdes finis. On peut appliquer ce qui précède au cas plus général des monoïdes finiment engendrés.

Définition 26 (Monoïdes finement engendrés)**.** *$M = \langle g \mid r \rangle$ avec g l'ensemble fini des générateurs et r un ensemble de relations entre générateurs.*

Exemple 22.
- *Monoïde libre : $\Sigma^* = \langle \Sigma \mid \emptyset \rangle$;*
- *Monoïdes libres partiellement commutatifs (ou monoïdes de traces) les relations sont des relations de commutation entre certaines paires de lettres : $M = \langle a, b, c \mid ab = ba \rangle$;*

Définition 27 (Ensemble rationnel)**.** *L'ensemble des rationnels d'un monoïde M est la plus petite classe de parties de M telles que*
- *$\forall g$ générateur, $g \in Rat(M)$;*
- *$\forall L, L' \in Rat(M)$, $(L \cup L') \in Rat(M)$;*

– $\forall L, L' \in Rat(M)$, $(L \cdot L') \in Rat(M)$;
– $\forall L \in Rat(M)$, $L^* \in Rat(M)$.

Exemple 23. *Considérons le monoïde* $\langle a, b \mid ab = ba \rangle$ *qui définit l'ensemble des mots sur l'alphabet* $\{a, b\}$ *pour lesquels deux mots ne différant que par une commutation de deux lettres successives sont équivalents. Partant de mots ayant le même nombre de a que de b (par exemple les éléments de* $(ab)^*$*, par commutations successives, on voit que l'on peut former les mots de la forme* $a^n b^n$ *qui représentent les classes d'équivalence de la relation de commutation. Donc, dans ce monoïde, on a* $L = (ab)^* = \{a^n b^n, n \in \mathbb{N}\}$*.* L *est rationnel.*

Est-il reconnaissable par un morphisme fini ? NON.

Dans ce cas, il faut distinguer les notions de rationalité (partie rationnelle d'un monoïde) et de reconnaissabilité (par un morphisme).

Chapitre 4

Grammaire et langages algébriques

4.1 Définitions et exemples

4.1.1 Définitions

Définition 1. *Une grammaire est un quadruplet $G = (\Sigma, V, S, P)$, où*
- Σ *est l'alphabet des* terminaux *;*
- V *est l'alphabet des* non-terminaux *(ou des* variables*) ;*
- $S \in V$ *est l'*axiome
- $P \subseteq (\Sigma \cup V)^* V (\Sigma \cup V)^* \times (\Sigma \cup V)^*$ *est un ensemble fini de règles de productions.*

Si $(L, R) \in P$, on note $L \to R$ et si $(L, R_1), (L, R_2) \in P$, on note $L \to R_1 | R_2$.

Exemple 1. *$G = (\Sigma, V, S, P)$ avec $\Sigma = \{a, b\}$, $V = \{S\}$, $P = \{S \to aSb \mid \epsilon\}$.*

Le principe de base des grammaires est de transformer des mots, et remplaçant le membre gauche d'une règle de production dans un mot par le membre droit : $S \to \epsilon$ ou $S \to aSb \to a(aSb)b \to aabb$.

Définition 2 (Dérivation)**.**
- *On dit que u* se dérive directement *en v (noté $u \to v$) si $u = u_1 L u_2$, $v = u_1 R u_2$ et $(L, R) \in P$;*
- *on dit que u* se dérive en n pas *en v (noté $u \to^n v$) s'il existe $u_0 = u, u_1 \dots u_{n-1}, u_n = v$ tels que $u \to u_1 \to u_2 \to \dots \to u_{n-1} \to v$;*
- *on dit que u* se dérive *en v (noté $u \to^* v$) si $u = v$ ou $\exists n \in \mathbb{N}$ tel que $u \to^n v$.*

Définition 3 (Langages engendrés)**.** *Le langage engendré par une grammaire $G = (\Sigma, V, S, P)$ est $L(G) = \{u \in \Sigma^* \mid S \to^* u\}$ et le langage étendu engendré par G est $\hat{L}(G) = \{u \in (\Sigma \cup V)^* \mid S \to^* u\}$.*

4.1.2 Retour sur la hiérarchie de Chomsky

Type 0 : engendré par une grammaire générale.

Type 1 : langages contextuels, engendrés par des grammaires contextuelles : $\forall(L,R) \in P$, $\exists g,d,m \in (\Sigma \cup V)^*$, $\exists A \in V$, tels que $L = gAd$ et $R = gmd$. g et d forment le contexte de la dérivation.

Exemple 2. $S \to abc|aTBc$
$TB \to TW$; $TW \to BW$; $BW \to BT$
$Tc \to XBcc$
$BX \to BY$; $BY \to XY$; $XY \to XB$
$aX \to aaT|aa$
$B \to b$
On génère ici le langage $L(G) = \{a^n b^n c^n, n \in \mathbb{N}^+\}$.
Par exemple, $S \to abc$ *ou* $S \to aTBc \to aTWc \to aBWc \to aBTc \to aBXBcc \to aBYBcc \to aXYBcc \to aXBBcc \to aaBBcc \to^* aabbcc$.

Type 2 : langages algébriques, grammaires algébriques (hors contexte) : $P \subseteq V \times (\Sigma \bigcup V)^*$.
Par exemple, $S \to aSb|\epsilon$, engendre le langage $L(G) = \{a^n b^n, n \in \mathbb{N}\}$.

Type 3 : grammaires linéaires à droite : $P \subseteq V \times (V\Sigma^* \bigcup \Sigma^*)$ ou grammaires linéraires à gauche : $P \subseteq V \times (\Sigma^* V \bigcup \Sigma^*)$.

Exemple 3.

$$\begin{aligned} S &\to aS \mid aT \\ T &\to bU \mid b \\ U &\to aU \mid b \end{aligned}$$

Le langage reconnu par cette grammaire est reconnaissable par automate fini : les non-terminaux sont les états (on ajoute un état final), et l'axiome S *est l'état initial.*

Plus généralement, les langages engendrés par une grammaire linéaire à droite ou linéaire à gauche sont eactement les langages rationnels.

4.2 Grammaires algébriques

$P \subseteq V \times (\Sigma \cup V)^*$
Un langage engendré par une grammaire algébrique est dit algébrique et on note $Alg(\Sigma)$ l'ensemble des langages algébriques sur Σ.

Le lemme fondamental suivant donne la conséquence de l'absence de contexte : les dérivations peuvent être conduites indépendamment sur les morceaux de la séquence courante.

4.2.1 Lemme fondamental

Lemme 1 (fondamental)**.** *Considérons* $u_1, u_2 \in (\Sigma \cup V)^*$*. Si* $u_1 u_2 \rightarrow^k v$ *alors* $\exists v_1, v_2 \in (\Sigma \cup V)^*, k_1, k_2 \in \mathbb{N}$ *tels que :* $u_1 \rightarrow^{k_1} v_1$*,* $u_2 \rightarrow^{k_2} v_2$*,* $k_1 + k_2 = k$ *et* $v_1 v_2 = v$*.*

Démonstration. On prouve ce lemme par récurrence :

- $k = 0$: on choisit $v_1 = u_1$, $v_2 = u_2$ et $k_1 = k_2 = 0$.
- Supposons la propriété établie pour k. Si $u_1 u_2 \rightarrow^{k+1} v$, alors $\exists w$ tel que $u_1 u_2 \rightarrow w \rightarrow^k v$. Il existe donc des mots $x, y \in (\Sigma \cup V)^*$, et une règle $T \rightarrow m \in P$ tels que $u_1 u_2 = xTy$ et $w = xmy$. Soit T apparaît dans u_1, soit il apparaît dans u_2. Dans le premier cas, x est un préfixe strict de u_1 et alors il existe $z \in (\Sigma \cup V)^*$ tel que $u_1 = xTz$. $u_1 \rightarrow xmz$ et donc $w = xmzu_2$. Dans l'autre cas, il existe z tel que $u_2 = zTy$ et, de la même manière $w = u_1 zmy$. Il existe donc d'après ce qui précède $w_1, w_2 \in (\Sigma \cup V)^*$ et $\ell_1, \ell_2 \in \mathbb{N}$ tels que $w = w_1 w_2$, $\ell_1 + \ell_2 = 1$, $u_1 \rightarrow^{\ell_1} w_1$ et $u_2 \rightarrow^{\ell_2} w_2$. On applique alors l'hypothèse de récurrence pour les mots w_1 et w_2. Il existe $v_1, v_2 \in (\Sigma \cup V)^*$ et $m_1, m_2 \in \mathbb{N}$ tels que $v = v_1 v_2$, $m_1 + m_2 = k$, $w_1 \rightarrow^{m_1} v_1$ et $w_2 \rightarrow^{m_2} v_2$. On a donc bien, finalement, l'existence de $k_1 = \ell_1 + m_1$, de $k_2 = \ell_2 + m_2$, de v_1 et v_2 tels que $k_1 + k_2 = k + 1$, $v = v_1 v_2$ et $u_1 \rightarrow^{k_1} v_1$ et $u_2 \rightarrow^{k_2} v_2$.

□

Ce lemme se généralise facilement.

Lemme 2 (fondamental généralisé)**.** *Si* $u_1 \dots u_n \rightarrow^k v$*, alors* $\exists v_1, \dots, v_m = v$ *et* $k_1, \dots, k_n$ *tels que* $\sum_{i=1}^n k_i = k$ *et* $\forall i$*,* $u_i \rightarrow^k v_i$*.*

4.2.2 Exemple : Langage de Lukasievicz

Alphabet : $\Sigma = \{a, b\}$
Grammaire G : $S \rightarrow aSS|b$

Considérons le langage L tel que :

1. $\forall u \in L, |u|_b = |u|_a + 1$
2. $\forall u \in L, \forall v$ préfixe strict de u, $|v|_a \geq |v|_b$

On veut montrer que $L = L(G)$.

Démonstration. Montrons tout d'abord par récurrence forte sur le nombre de dérivations que $L(G) \subseteq L$:

$$HR(k) \quad : \quad S \rightarrow^k u \Rightarrow u \in L.$$

$HR(1)$: La seule dérivation possible est $S \rightarrow b$. Et on a bien $b \in L$.

Supposons maintenant $HR(\ell)$ vérifiée pour tout $\ell \in \{1, \dots, k\}$ et montrons que $HR(k+1)$ est vérifiée.

On a $S \rightarrow^{k+1} u$ donc $S \rightarrow aSS \rightarrow^{k} u$. D'après le lemme fondamental, il existe $u_1, u_2 \in \Sigma^*$ tels que $u = au_1u_2$ avec $S \rightarrow^{\ell_1} u_1$, $S \rightarrow^{\ell_2} u_2$, $\ell_1 + \ell_2 = k$. En particulier $\ell_1 \leq k$ et $\ell_2 \leq k$, donc par hypothèse de récurrence $u_1, u_2 \in L$. On a donc

$$|u|_a = 1 + |u_1|_a + |u_2|_a = 1 + |u_1|_b - 1 + |u_2|_b - 1 = |u|_b - 1$$

u vérifie donc le premier item de la définition de L. Un préfixe strict de u est de la forme av_1 avec v_1 préfixe de u_1 ou au_1v_1 avec v_1 préfixe strict de u_2. Pour le premier cas, $|v|_a = |v_1|_a + 1$. Comme $u_1 \in L$, $|v_1|_a = |v_1|_b - 1$, $|v|_a = |v_1|_b = |v|_b$. Ce qui permet de conclure. Dans l'autre cas, $v = au_1v_1$. Comme v_1 est préfixe strict de u_2 et $u_2 \in L$, on a $|v_1|_a \geq |v_1|_b$. Donc $|v|_a = |u_1|_a + |v_1|_a + 1 = |u_1|_b + |v_1|_a \geq \|u_1|_b + |v_1|_b = |v|_b$.

Montrons maintenant l'autre implication : $L \subseteq L(G)$. Soit u un mot de L. Montrons par récurrence forte sur la longueur de u que $u \in L \Rightarrow u \in L(G)$.

$$HR'(k) \quad : \quad u \in L \text{ et } |u| = k \Rightarrow S \rightarrow^* u.$$

$HR'(1)$: la seule possibilité est $u = b$. On a bien $S \rightarrow b$.

Supposons $HR'(\ell)$ vérifiée pour tous $\ell \in \{1, \ldots, k\}$ et montrons que $HR'(k+1)$ est aussi vérifiée.

Soit $u \in L$ tel que $|u| = k + 1$. Ce mot commence nécessairement par a. Soit u_1 le plus petit mot tel que au_1 est préfixe de u et $|au_1|_a = |au_1|_b$. On a $|u_1|_a = |u_1|_b - 1$. Et pour tout préfixe av_1 de au_1 on a $|av_1|_a \geq |av_1|_b$ (av_1 est préfixe strict de u) donc $|v_1|_a \geq |v_1|_b$. De plus, comme $|au_1|_a = |au_1|_b$, on a $|u_1|_a + 1 = |u_1|_b$. Donc $u_1 \in L$.

Soit le mot u_2 tel que $u = au_1u_2$.

$$|u_2|_a = |u|_a - |au_1|_a = |u|_b - 1 - |au_1|_b = |u_2|_b - 1$$

et pour tout v_2 préfixe strict de u_2,

$$|v_2|_a = |au_1v_2|_a - |au_1|_a \geq |au_1v_2|_b - |au_1|_b = |v_2|_b.$$

Donc $u_2 \in L$.

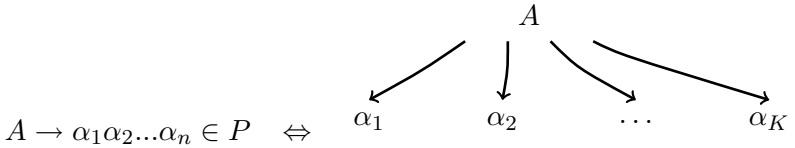
Donc on a $S \rightarrow^* u_1$ et $S \rightarrow^* u_2$, et u est engendré par la grammaire G : $S \rightarrow aSS \rightarrow^* au_1u_2 = u$.

□

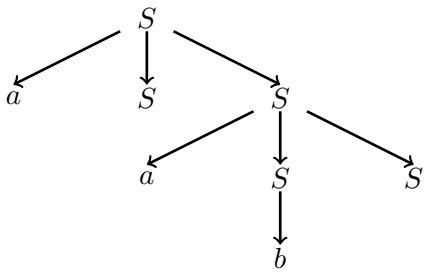
4.2.3 Arbres de dérivation

Reprenons la grammaire de l'exemple précédent : $S \rightarrow aSS|b$. Plusieurs dérivations engendrent le même mot : $S \rightarrow aSS \rightarrow aaSSS \rightarrow aabSS \rightarrow aabbS \rightarrow aabbb$ et $S \rightarrow aSS \rightarrow aSb \rightarrow aaSSb \rightarrow aaSbb \rightarrow aabbb$. En regardant ces dérivations de plus près, on se rend compte que seul l'ordre des dérivations change, mais les règles de production utilisées restent les mêmes. L'arbre de dérivation permet d'identifier les dérivations qui ne diffèrent que par l'ordre des dérivations.

Définition 4. *Un arbre syntaxique est un arbre dont*
- *les sommets sont étiquetés par des lettres de* $\Sigma \cup V$
- *les racines sont étiquetées par un non terminal*
-



Définition 5 (Mot de feuille)**.** *mot formé par les feuilles lus de gauche à droite.*



Exemple 4.

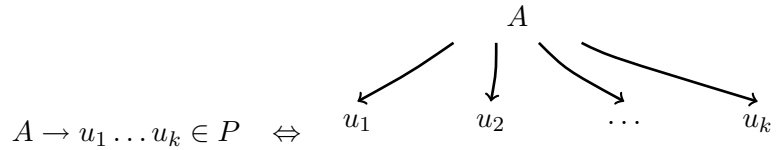
Le mot de feuille est aSabS.

Définition 6. *Un arbre de dérivation est un arbre syntaxique dont :*
- *la racine est l'axiome S*
- *les feuilles sont des lettres terminales*

Théorème 1. *Soit G une grammaire algébrique*
$u \in L(G) \Leftrightarrow u$ *est un mot de feuille d'un arbre de dérivation de G.*

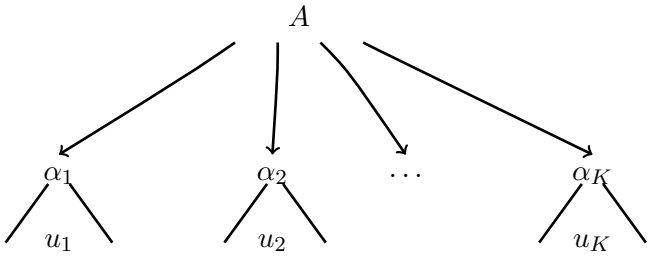
Démonstration. Soit $u \in L(G)$ tel que $S \to^n u$. Montrons par récurrence forte sur n que si $A \to^n u$ alors il existe un arbre syntaxique de racine A et de mot de feuille u .

Cas $n = 1$: on a par définition



Supposons la propriété vraie pour toutes les dérivations de longueur au plus n. Soit $u \in \Sigma^*$ tel que $A \to^{n+1} u$. On a alors $A \to \alpha_1...\alpha_k \to^n u$ et d'après le lemme fondamental, $\exists u_1, ..., u_k$ tel que $\alpha_i \to^{\ell_i} u$ avec $\sum \ell_i = n$. En particulier $\ell_i \leq n$.

Par l'hypothèse de recurrence $\forall i \in [|1, k|]$ il existe un arbre syntaxique de racine α_i et de mot de feuille u_i.

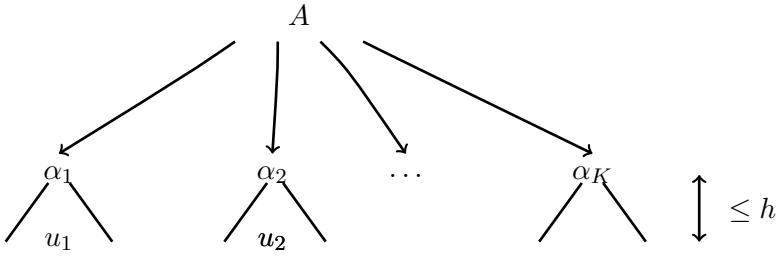


le mot de feuille est $u_1 ... u_k$, d'où le résultat en prenant $A = S$.

Montrons la réciproque par récurrence forte sur la hauteur h de l'arbre. Soit u un mot de feuille d'un arbre syntaxique.

Cas $h = 1$: idem cas $n = 1$ pour l'autre sens (on a une équivalence).

Supposons la propriété vraie pour tous les arbres de hauteur au plus h. Considérons un arbre de hauteur $h + 1$.



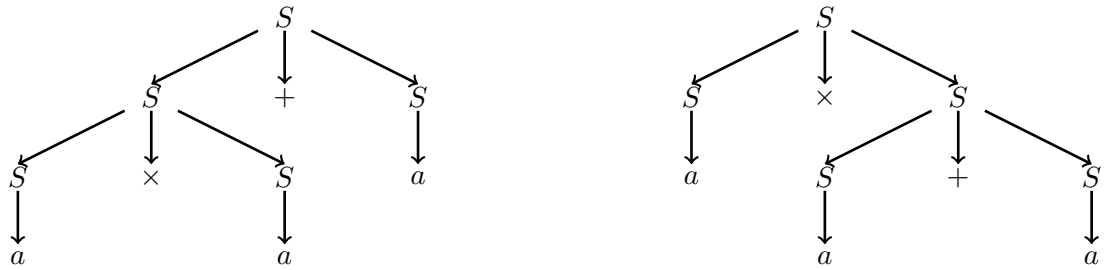
Par hypothèse de récurrence, pour tout i, $\alpha_i \to^* u_i$ et on a donc la dérivation $A \to \alpha_1 \dots \alpha_K \to^* u_1 \dots u_K = u$. □

Remarque : Les dérivations qui ont le même arbre de dérivation sont dites équivalentes. Un représentant particulier peut être la dérivation la plus à gauche (*i.e.* : on dérive toujours le non terminal le plus à gauche en premier).

4.2.4 Grammaires ambigües

Un mot peut avoir plusieurs arbres de dérivation.

Exemple 5. *Avec la grammaire $S \to S + S \mid S \times S \mid a$, on considère le mot $a \times a + a$.*



On peut choisir $S \to a \times S \mid a + S \mid a$ qui engendre le même langage mais qui n'est pas ambigüe.

Théorème 2. *Le problème de savoir si une grammaire algébrique est ambigüe est indécidable.*

Démonstration. Via le problème de correspondance de Post.

Problème de correspondance de Post : Soient $(u_1, v_1)...(u_n, v_n)$ n paires de mot. On veut savoir s'il existe $i_q, ..., i_m$ des indices (pas forcément distincts) tels que

$$u_{i_1}...u_{i_m} = v_{i_1}...v_{i_m}.$$

Soient $a_1, ...a_n \notin \Sigma$. La grammaire

$$\begin{aligned} A &\rightarrow u_1Aa_1 \mid u_2Aa_2 \mid \ldots \mid u_nAa_n \mid u_1a_1 \mid u_2a_2 \mid \ldots \mid u_na_n \\ B &\rightarrow v_1Ba_1 \mid v_2Ba_2 \mid \ldots \mid v_nBa_n \mid v_1a_1 \mid v_2a_2 \mid \ldots \mid v_na_n \\ S &\rightarrow A \mid B \end{aligned}$$

est ambigüe si on peut engendrer le même mot par A et B, *i.e.* s'il existe :

$$\begin{aligned} S &\rightarrow A \rightarrow^* wa_{i_n} \ldots a_{i_1} \\ S &\rightarrow B \rightarrow^* wa_{i_n} \ldots a_{i_1} \end{aligned}$$

alors $i_1 \ldots i_n$ est une solution au problème de correspondance de Post. Si la grammaire est ambigüe alors il n'y a pas de solution au problème de correspondance de Post. □

Un langage est intrinsèquement ambigu si toutes les grammaires algébriques qui l'engendrent sont ambigües.

Exemple 6. $L = \{a^n b^n c^m d^m,\ n, m \in \mathbb{N}\} \cup \{a^n b^m c^m d^n,\ n.n \in \mathbb{N}\}$ *est intrinsèquement ambigu.*

4.3 Lemme de l'étoile pour les grammaires algébriques

4.3.1 Nettoyage d'une grammaire

Le même langage pouvant être défini par des grammaires différentes, il est naturel d'essayer de simplifier leur description. Une étape facile est d'enlever les symboles non terminaux inutiles qui n'apparaissent pas dans les dérivations.

Définition 7. – *Non terminaux utiles $X \in V$ tel que*

$$\underbrace{S \rightarrow^* \alpha X \beta}_{X \text{ accessible}} \qquad \underbrace{\alpha X \beta \rightarrow^* u}_{X \text{ générateur}};$$

– *symboles inutiles : les autres non terminaux.*

Les symboles générateurs sont les symboles dans $U \setminus \Sigma$ où U est défini de la facon suivante :

$$\begin{cases} U_0 & = & \Sigma \\ U_{n+1} & = & \{A \in V \mid \exists A \to \alpha_1 \dots \alpha_k \text{ et } \forall i \in \{1, \dots, k\},\ \alpha_i \in U_n\} \cup U_n. \end{cases}$$

La suite U_n étant croissante pour l'inclusion et le nombre d'éléments de $\Sigma \cup V$ étant fini, cette suite est ultimement constante. On note $U = \cup_{n \in \mathbb{N}} U_n$.

Les symboles accessibles sont les symboles de $\cup_{n \in \mathbb{N}} V_n$ avec

$$\begin{cases} V_0 & = & \{S\} \\ V_{n+1} & = & \{A \in V \mid \exists B \in V_n,\ (B \to \alpha A \beta) \in P\} \cup V_n. \end{cases}$$

Élimination des symboles inutiles

1. éliminer les symboles non générateurs
2. éliminer les symboles non accessibles

Exemple 7.

$$S \to AB \mid a \qquad A \to b$$

Les symboles générateurs sont : S et A. Après avoir enlevé les non générateurs on obtient :

$$S \to a \qquad A \to b$$

Le symbole A devient donc non-accessible et on obtient $S \to a$.

On appelle *grammaire réduite* la grammaire obtenue après élimination des symboles inutiles.

Proposition 1. *Soit G une grammaire algébrique et G' sa grammaire réduite. Alors G et G' engendrent le même langage.*

Pour des raisons techniques, on veut éliminer les règles de production de type $A \to \varepsilon$.

Après cette opération de nettoyage, la grammaire engendre le même langage au mot vide près (qui ne peut plus être engendré par la grammaire propre).

On remarque d'abord qu'il ne faut supprimer simplement les règles $A \to \varepsilon$. En effet la grammaire

$$S \to aAa, A \to \varepsilon, A \to bA, A \to c$$

ne donnerait plus le mot aa si on enlevait la règle.

On cherche d'abord les non-terminaux A tels que $A \rightarrow^* \varepsilon$. ce sont les symboles de $U = \cup_{n\geq 0} U_n$ avec

$$\begin{cases} U_0 & = & \{A \in V | (A \rightarrow \varepsilon) \in P\} \\ U_{n+1} & = & \{A \in V | \exists (A \rightarrow \alpha_1 \dots \alpha_k) \in P \text{ et } \forall i \in \{1, \dots, k\}, \alpha_i \in U_n\} \cup U_n. \end{cases}$$

Ensuite, pour toutes les règles $A \rightarrow \alpha_i \dots \alpha_k$ avec $\alpha_1, \dots, \alpha_k \in V \cup \Sigma$, on ajoute toutes les règles possibles $A \rightarrow \alpha'_1, \dots, \alpha'_k$ avec

$$\alpha'_i = \begin{cases} \alpha_i & \text{si } \alpha_i \notin U, \\ \alpha_i \text{ ou } \varepsilon & \text{si } \alpha_i \in U. \end{cases}$$

et on élimine les règles de type $A \rightarrow \varepsilon$.

Exemple 8. *Sur l'exemple précédent, $U = \{A\}$. La grammaire transformée est :*

$$S \rightarrow aAa, S \rightarrow aa, A \rightarrow bA, A \rightarrow b, A \rightarrow c$$

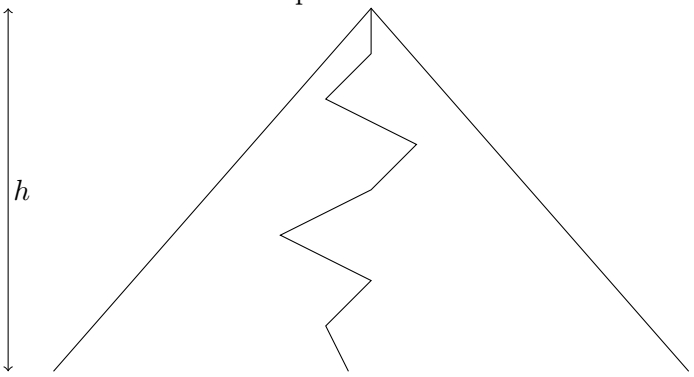
4.3.2 Lemme de l'étoile

Lemme 3. *Si L est un langage algébrique alors $\exists n \in \mathbb{N}, \forall u \in L$ tel que $|u| \geq n$ il existe une décomposition de $u = xvywz$ tel que*

$$\begin{cases} vw \neq \epsilon \\ |vyw| \leq n \\ \forall i \in \mathbb{N}, xv^i yw^i z \in L \end{cases}$$

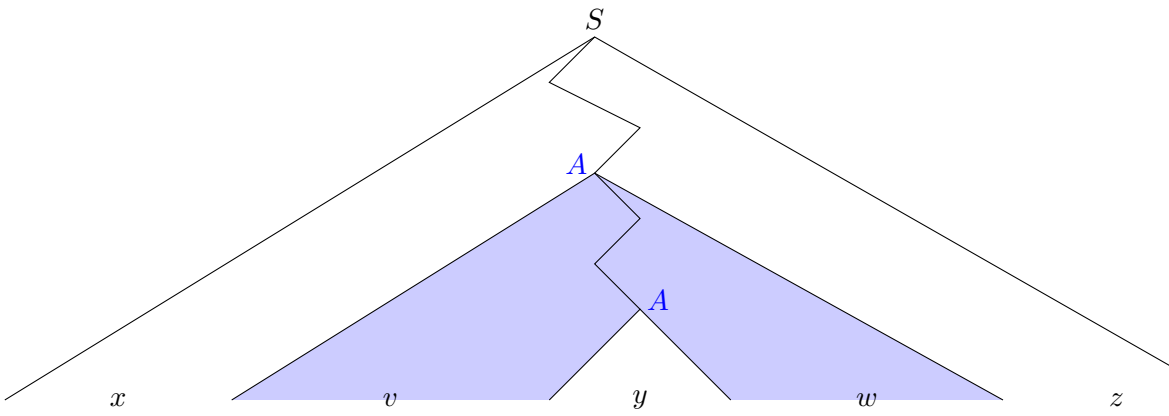
Démonstration. $m = max\{|\alpha| \mid A \rightarrow \alpha \in P\}$

Chaque nœud de l'arbre a au plus m fils. Soit un arbre de dérivation de hauteur h. Cet arbre a au plus m^h feuilles.



On pose $n = m^{|V|+1}$.

Les mots de longueur $\geq n$ ont des arbres de dérivation de hauteur au moins égale à $|V| + 1$. Pour un arbre de hauteur $\geq |V| + 1$, il existe un embranchement de hauteur au moins $|V| + 1$ avec $|V| + 1$ non terminaux. Donc $\exists A \in V$ qui apparaît deux fois sur cet embranchement.



$S \to^* xAz$
$A \to^* vAw \mid y$
$S \to^* xv^iAw^iz \to^* xv^iyw^iz$ □

Application : $L = \{a^nb^nc^n,\ n \in \mathbb{N}\}$ n'est pas algébrique

On suppose que L est algébrique. Le lemme fournit un entier n. Si v de ce lemme n'est pas du type a^k,b^k ou c^k alors les itérations sortent de $a^*b^*c^* \supset L$. De même, w est de la forme a^ℓ,b^ℓ ou c^ℓ. De plus, comme $vw \neq \epsilon$, k ou ℓ n'est pas nul. Par itérations, on augmente donc le nombre de une ou deux lettres mais jamais des trois. Donc L ne vérifie pas le lemme et n'est donc pas algébrique.

Remarque : les langages algébriques ne sont pas stables par intersection $\{a^nb^nc^n,\ n \in \mathbb{N}\} = \{a^mb^nc^n,\ m,n \in \mathbb{N}\} \cap \{a^nb^nc^m,\ m,n \in \mathbb{N}\}$. On sait que les deux langages considérés sont algébriques ($\{a^nb^nc^m,\ m,n \in \mathbb{N}\}$ est engendré par la grammaire $S \to S_1S_2, S_1 \to aS_1b|ab, S_2 \to cS_2|c$).

4.4 Formes normales de grammaire

4.4.1 Forme normale de Chomsky (forme normale quadratique, CNF)

On ne considère que les règles de la forme $A \to a$ ou $A \to BC$.

Théorème 3. *Toute grammaire algébrique peut être transformée en CNF qui engendre le même langage.*

Démonstration. cf TD □

Application : algorithme CYK (Cocke, Yanger, Kamada)

Algorithme décidant si un mot u peut être engendré par une grammaire donnée en CNF.

idée : $u = u_1u_2...u_n$ à partir de quels non terminaux peut-on engendrer u ?

Si $A \rightarrow^* u_1...u_n$
- soit $n = 1$ et $A \rightarrow u_1 = u$
- soit $\exists B, C \in V$, $\exists k \in \{1, 2, ..., n\}$ tels que $A \rightarrow BC$, $B \rightarrow^* u_1...u_k$ et $C \rightarrow^* u_{k+1}...u_n$.

On note $X_{i,j} = \{A \in V, A \rightarrow^* u_i...u_j\}$, l'ensemble des non-terminaux qui engendrent le facteur $u_i \dots u_j$.

On calcule les $X_{i,j}$ de manière récursive :
- $X_{i,i} = \{A \in V \mid A \rightarrow u_i \in P\}$
- $X_{i,j} = \{A \in V \mid \exists B, C \in V, \exists k \in \{1, 2, ..., n\}, A \rightarrow BC,\ B \in X_{i,k} \text{ et } C \in X_{k+1,j}\}$

Finalement, $u \in L \Leftrightarrow S \in X_{1,n}$

Algorithme 1 : CYK(u)

```
pour i = 1 à n faire
    X_{i,i} ← {A ∈ V | A → u_i ∈ P}
pour ℓ = 1 à n − 1 faire
    pour i = 1 à n − l faire
        X_{i,i+l} ← ∅
        pour k = 1 à ℓ − 1 faire
            X_{i,i+l} ← X_{i,i+l} ∪ {A ∈ V | ∃B ∈ X_{i,i+k}, ∃C ∈ X_{k+1,ℓ}, A →
            BC}
si S ∈ X_{1,n} alors
    u ∈ L(G)
sinon
    u ∉ L(G)
```

Exemple 9. $S \rightarrow AB \mid BC$
$A \rightarrow BA \mid a$
$B \rightarrow CC \mid b$
$C \rightarrow AB \mid a$
$u = baaba$ est-il engendré par cette grammaire CNF G ?

	b	a	a	b	a
$X_{i,i}$	$\{B\}$	$\{A, C\}$	$\{A, C\}$	$\{B\}$	$\{A, C\}$
$X_{i,i+1}$	$\{A, S\}$	$\{B\}$	$\{S, C\}$	$\{A, S\}$	$\times$
$X_{i,i+2}$	$\varnothing$	$\{B\}$	$\{B\}$	$\times$	$\times$
$X_{i,i+3}$	$\varnothing$	$\{S, C, A\}$	$\times$	$\times$	$\times$
$X_{i,i+4}$	$\{A, S; , \}$	$\times$	$\times$	$\times$	$\times$

$S \in X_{1,5}$ *donc* $u \in L(G)$.

On pourrait de plus retenir la construction qui amène S à la dernière ligne, ce qui permettrait de connaître une dérivation donnant u à partir de S.

$S \to BC \to bC \to bAB \to baB \to baCC \to baCa \to baABa \to baaBa \to baaba$.

Données de l'algorithme : $u \in \Sigma^*$
Question : $u \in ?L(G)$
Complexité : $O(|u|^3)$

Application : théorème de Chomsky Schutzenberger

Résultats préalables :

1. Langages de Dyck : soit $\Sigma = \{a_1, ..., a_n, \bar{a}_1, ..., \bar{a}_n\}$. Le langage de Dyck $D_n^* = \{$ mots bien parenthésés à n types de parenthèses $\}$ engendré par $\forall i \in \{1, ..., n\},\ S \to a_i S \bar{a}_i S \mid \epsilon$.
2. Stabilité des langages algébriques :
   - par morphisme : $\varphi : \Sigma^* \to \Sigma'^*, L$ algébrique $\Rightarrow \varphi(L)$ algébrique (les règles $A \to a$ se transforment en $A \to \varphi(a)$, qui elles mêmes peuvent se transformer en des règles algébriques pouvant former le mot $\varphi(a)$).
   - par intersection avec un rationnel : L algébrique, R régulier $\Rightarrow L \cap R$ algébrique.

Théorème 4. *Un langage L est algébrique si et seulement si :*
- $\exists$ *un morphisme alphabétique* φ
- $\exists n \in \mathbb{N}$
- $\exists R \in Rat(\Sigma')$

tels que $L = \varphi(D_n^* \cap R)$.

Un morphisme alphabétique est un morphisme tel que :
$\forall a \in \Sigma, |\varphi(a)| \leq 1$.

Démonstration. $\Leftarrow$ se déduit facilement par la stabilité des langages algébriques (D_n^* en est un) par intersection avec un langage rationnel et par morphisme.

$\Rightarrow$ Soit $G = (\Sigma, V, S, P)$ une forme normale de Chomsky engendrant L. On rappelle que l'on a alors deux types de règles :
- $T \to T_1 T_2$ (ℓ règles)
- $T \to a$ (m règles)

Pour chaque règle r, on crée les nouveaux symboles suivants, qui vont constituer un alphabet pour un langage de Dyck D_n :
- $a_r, b_r, c_r, \bar{a}_r, \bar{b}_r, \bar{c}_r$ si $r : T \to T_1 T_2$
- $a_r, \bar{a}_r$ si $r : T \to a$

$\hookrightarrow n = 3\ell + m$.

On transforme la grammaire G en G' par :

$$\begin{aligned} r : T \to T_1 T_2 &\rightsquigarrow T \to a_r b_r T_1 \bar{b_r} c_r T_2 \bar{c_r} \bar{a_r} \\ r : T \to a &\rightsquigarrow T \to a_r \bar{a_r}. \end{aligned}$$

On choisit ensuite le morphisme alphabétique

$$\begin{aligned}\varphi(a_r) = \varphi(\bar{a_r}) = \varphi(b_r) = \varphi(\bar{b_r}) = \varphi(c_r) = \varphi(\bar{c_r}) = \epsilon \qquad & \text{si } r : T \to T_1T_2 \\ \varphi(a_r) = a, \varphi(\bar{a_r}) = \epsilon \qquad & \text{si } r : T \to a.\end{aligned}$$

On a $\varphi(L') = L$. Le langage L' est bien parenthésé et donc $L' \subseteq D_n^*$. Il reste à définir R.

On exprime les contraintes sur les mots de L' pour deux lettres consécutives. On note K l'ensemble des facteurs de longueur 2 autorisés.

$$K = \begin{cases} a_r\bar{a}_r & \text{si } r : T \to a \\ a_rb_r,\ \bar{b_r}c_r,\ \bar{c}_r\bar{a}_r & \text{si } r : T \to T_1T_2 \\ b_ra_{r'},\ \bar{a}_{r'}\bar{b}_r & \text{si } r : T \to T_1T_2 \text{ et } r' : T_1 \to ... \\ c_ra_{r'},\ \bar{a}_{r'}\bar{c}_r & \text{si } r : T \to T_1T_2 \text{ et } r' : T_2 \to ... \end{cases}$$

On pose $R = \Sigma_0\Sigma^* \setminus \Sigma^*(\Sigma^2 \setminus K)\Sigma^*$ avec $\Sigma_0 = \{a_r, r : S \to ...\}$.

On déduit de ces contraintes que $L' \subseteq D_n^* \cap R$.

Pour finir la preuve, on montre que par récurrence que $D_n^* \cap R \subseteq L'$.

On pose $\Sigma_T = \{a_r, r : T \to ...\}$ et $R_T = \Sigma_T\Sigma^* \setminus \Sigma^*(\Sigma^2 \setminus K)\Sigma^*$

Hypothèse de récurrence :

$$\forall T \in V,\ \forall u \in D_n^*,\ u \in D_n^* \cap R_T \Rightarrow u \in L(G', T).$$

Pour $k = 2 : u = a_r\bar{a_r}$ et $r : T \to a$.

Supposons l'hypothèse de récurrence vérifiée jusqu'à l'ordre k et montrons qu'elle est alors vérifiée à l'ordre $k + 1 > 2$.

Soit $|u| = k + 1 > 2$ et supposons que $u \in \Sigma_T\Sigma^*$. On a nécessairement u de la forme

$$u = a_rb_rv_1\bar{b}_rc_rv_2\bar{c}_r\bar{a}_r.$$

où v_1 et v_2 sont bien parenthésés. Les seules lettres qui peuvent suivre $\bar{a}_r$ sont des lettres de type $\bar{b}$ ou $\bar{c}$. ce n'est pas possible, puisque les mots doivent être bien parenthèsés. Donc $\bar{a}_r$ est donc bien la dernière lettre du mot u.

La règle r est $T \to T_1T_2$. Donc nécessairement, v_1 commence par $a_{r'}$ tel que $r' : T_1 \to ...$ et v_2 commence par $a_{r''}$ tel que $r'' : T_2 \to ....$ Par suite, $v_1 \in D_n^* \cap R_{T_1}$ et $v_2 \in D_n^* \cap R_{T_2}$ et $v_1 \in L(G', T_1)$ et $v_2 \in L(G', T_2)$ par hypothèse de récurrence.

On a la dérivation suivante :

$$T \to a_rb_rT_1\bar{b_r}c_rT_2\bar{c_r}a_r \to^* a_rb_rv_1\bar{b}_rc_rv_2\bar{c}_r\bar{a}_r = u,$$

et $T \to^* u \in L(G', T)$ □

4.4.2 Forme normale de Greibach

Définition 8. *Une grammaire* $G = (\Sigma, V, S, P)$ *est en* forme normale de Greibach *(GNF) si ses règles de production sont de la forme :*

$$A \to a\alpha \ \ (a \in \Sigma, \alpha \in V^*).$$

Proposition 2. *Toute grammaire algébrique ne contenant pas le mot vide peut être transformée en une grammaire GNF qui engendre exactement le même langage.*

Remarque :

1. Si $A \to \alpha B\gamma$, avec $A, B \in V,\ \alpha, \gamma \in (\Sigma \cup V)^*$ et si $B \to \beta_1 \mid \ldots \mid \beta_k$ désigne toutes les règles avec B à gauche, on peut remplacer $A \to \alpha B\gamma$ par $A \to \alpha\beta_1\gamma \mid \ldots \mid \alpha\beta_k\gamma$ sans modifier le langage engendré.
2. Si $A \to A\alpha_1 \mid \ldots \mid A\alpha_\ell \mid \beta_1 \mid \ldots \mid \beta_k$ désigne l'ensemble des règles avec A à gauche avec $\beta_i \notin A(\Sigma \cup V)^*$, on peut remplacer $A \to A\alpha_1 \mid \ldots \mid A\alpha_l$ par :

   $A \to \beta_1 \mid \ldots \mid \beta_k \mid \beta_1 Z \mid \ldots \mid \beta_k Z$
   $Z \to \alpha_1 \mid \ldots \mid \alpha_\ell \mid \alpha_1 Z \mid \ldots \mid \alpha_l Z$ $\ Z$, nouvelle variable de la grammaire

   sans modifier le langage engendré.

Démonstration. On part d'une grammaire en CNF. Les règles sont de la forme $A \to BC$ et $A \to a$. On énumère les variables $V = \{A_1, \ldots, A_n\}$.

1ère étape On commence par transformer les règles $A_i \to A_j\alpha$ avec $j \leq i$ pour ne plus garder que les règles $A_i \to A_j\alpha$ avec $j > i$ ($\alpha \in V$). On procède par induction.

Pour A_1 : On peut écrire toutes les règles avec A_1 à gauche comme $A_1 \to A_1\alpha_1 \mid \ldots \mid A_1\alpha_\ell \mid \beta_1 \mid \ldots \mid \beta_k$, avec $\beta_j = a$ ou A_jA_k, avec $j > 1$.

On applique la transformation de la Remarque 2 en introduisant A_{-1} :

$$A_1 \to \beta_1 \mid \ldots \mid \beta_k \mid \beta_1 A_{-1} \mid \ldots \mid \beta_k A_{-1}$$

$$A_{-1} \to \alpha_1 \mid \ldots \mid \alpha_\ell \mid \alpha_1 A_{-1} \mid \ldots \mid \alpha_\ell A_{-1}$$

On suppose maintenant la transformation effectuée jusqu'à l'ordre k, et on le fait à l'ordre $k+1$.

On regarde les règles de type $A_{k+1} \to A_j\alpha$.

- Si $A_{k+1} \to A_j\alpha$ avec $j < k+1$, alors on applique la remarque 1 puisque les règles $A_j \to A_\ell\beta$ sont telles que $\ell > j$. En appliquant cette remarque 1 tant qu'il existe des règles $A_{k+1} \to A_j\alpha$ avec $j < k+1$, on obtient une grammaire avec seulement des règles de type $A_{k+1} \to A_j\alpha$ avec $j \geq k+1$. (Cette construction termine nécessairement).
- Si pour toutes les règles on a $A_{k+1} \to A_j\alpha$ avec $j > k+1$ alors la transformation est terminée.
- Sinon, on applique la même transformation que pour le cas $k = 1$ en introduisant un nouvel état non terminal A_{-k-1}.

2ème étape Transformation en GNF.

On applique successivement la transformation de la remarque 1.

- les règles avec A_n à gauche sont nécessairement de la forme $A_n \to a\alpha$, $\alpha \in V^*$, donc pas de transformation à effectuer.
- Pour A_{n-1}, des règles $A_{n-1} \to A_n\alpha$ peuvent exister, et on applique alors la remarque 1 pour le remplacement de A_n.
- On continue comme cela pour tous les non-terminaux.

□

Exemple 10.

$$\begin{aligned} A &\to BA \mid a \\ B &\to CC \mid b \\ C &\to AB \mid a \end{aligned}$$

1ère étape :

$$\begin{aligned} A &\to BA \mid a \\ B &\to CC \mid b \\ C &\to AB \mid a \\ C &\to BAB \mid aB \mid a \\ C &\to C\overbrace{CAB}^{\alpha_1} \mid bAB \mid aB \mid a \\ C &\to bAB \mid aB \mid a \mid bAB\overline{C} \mid aB\overline{C} \mid a\overline{C} \\ \overline{C} &\to CAB \mid CAB\overline{C} \end{aligned}$$

2ème étape :

$$\begin{aligned} C &\to bAB \mid aB \mid a \mid bAB\overline{C} \mid aB\overline{C} \mid a\overline{C} \\ B &\to bABC \mid aBC \mid aC \mid bAB\overline{C}C \mid aB\overline{C}C \mid a\overline{C}C \mid b \\ A &\to \cdots \\ \overline{C} &\to \cdots \end{aligned}$$

Chapitre 5

Automates à pile

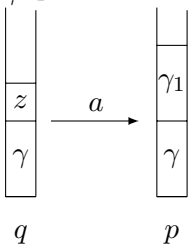
5.1 Définition

Définition 9. *Un automate à pile est un 7-uplet :* $\mathcal{A} = (Q, \Sigma, \Gamma, \delta, q_0, \gamma_0, F)$

- Q *: ensemble fini d'états.*
- Σ *: alphabet fini (terminal).*
- Γ *: alphabet de pile.*
- $\delta : (Q \times \Sigma \cup \{\epsilon\} \times \Gamma) \to \mathcal{P}_{finie}(Q \times \Gamma^*)$ *fonction de transition.*
- γ_0 *: symbole initial de fond de pile*
- $q_0 \in Q$ *état initial.*
- $F \subseteq Q$ *états terminaux.*

5.1.1 Configuration :

- $(q, \gamma) \in Q \times \Gamma^*$.
- (q_0, γ_0) configuration initiale.
- $(q, \gamma z) \xrightarrow{a} (p, \gamma\gamma_1)$ ssi $(p, \gamma_1) \in \delta(q, a, z)$, avec $a \in \Sigma \cup \{\varepsilon\}$, $z \in \Gamma$ et $\gamma \in \Gamma^*$.



5.1.2 Lecture

On ne peut lire une lettre et changer d'état que si la pile est non vide.

- $(q, \gamma) \xrightarrow{u\ *} (p, \gamma')$ s'il existe $n \in \mathbb{N}$ et $(q_0, \gamma_0) = (q, \gamma), (q_1, \gamma_1), \ldots, (q_n, \gamma_n) = (p, \gamma')$ tels que $\forall i \in [1, \ldots, n]$, $(q_{i-1}, \gamma_{i-1}) \xrightarrow{u_i} (q_i, \gamma_i)$ et $u = u_1 \ldots u_n$, avec $u_i \in \Sigma \cup \{\varepsilon\}$.

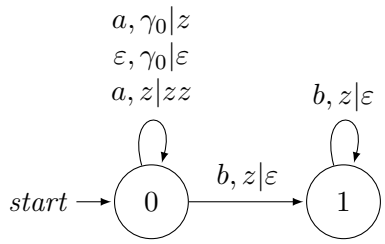
2 modes de reconnaissances

- par pile vide :
  $L_p(\mathcal{A}) = \{u \in \Sigma^* \mid (q_0, \gamma_0) \xrightarrow{u\ *} (q_1, \epsilon),\ q_1 \text{ quelconque}\}$.
- par état terminal :
  $L_f(\mathcal{A}) = \{u \in \Sigma^* \mid \exists q \in F\ (q_0, \gamma_0) \xrightarrow{u\ *} (q, \gamma)\}$.

Exemple 11.

$$(0, \gamma_0) \xrightarrow{\epsilon} (0, \epsilon)$$

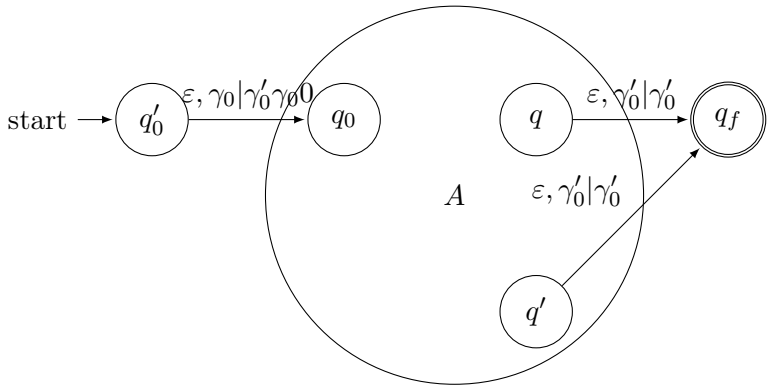
$$(0, \gamma_0) \xrightarrow{a} (0, z) \xrightarrow{a} (0, zz) \xrightarrow{b} (1, z)$$



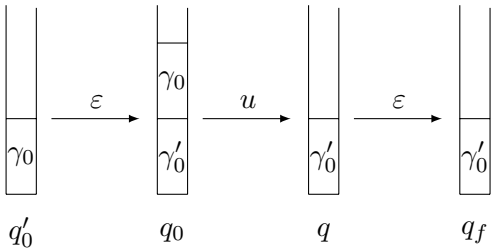
En essayant de construire par simulation exhaustive l'automate des états, on reconnaît l'automate (à nombre d'états infini) engendrant le langage algébrique $\{a^n b^n | n \in \mathbb{N}\}$.

Proposition 3. *Tout langage reconnaissable par un automate à pile par pile vide est reconnaissable par un automate à pile par état terminal et inversement.*

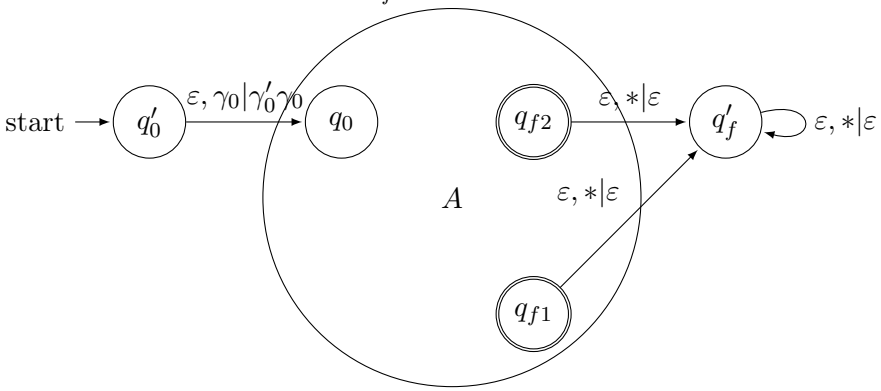
Démonstration. Soit L un langage reconnu par $A = (Q, \Sigma, \Gamma, \delta, q_0, \gamma_0, F)$ un automate à pile par pile vide. On rajoute alors un symbole γ_0' dans l'alphabet de pile, un état initial q_0' et un état final q_f.



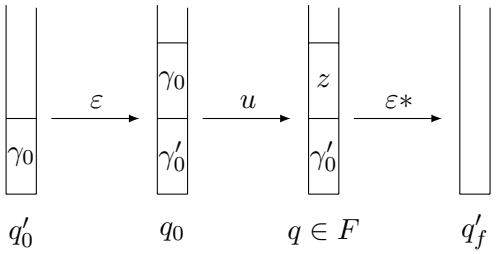
Evolution de la pile pour un mot u reconnu par A :



Soit L un langage reconnu par $A = (Q, \Sigma, \Gamma, \delta, q_0, \gamma_0, F)$ un automate à pile par état terminal. On rajoute alors un symbole γ_0' dans l'alphabet de pile, un état initial q_0' et un état q_f' pour vider la pile.



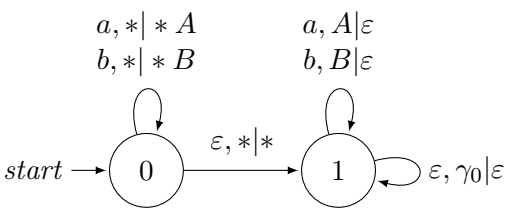
Evolution de la pile pour un mot u reconnu par A :



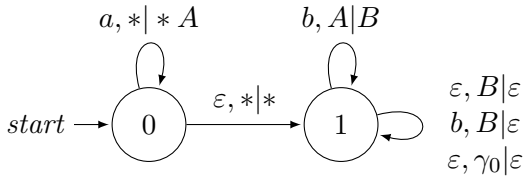
avec $z \in \Gamma^*$.

□

Exemple 12. *1. L'automate ci-dessous reconnaît par pile vide les palindromes pairs sur l'alphabet $\{a, b\}$. L'idée est de recopier dans une première phase le mot d'entrée dans la pile, puis dans une deuxième phase de dépiler ce mot. On obtient ainsi un mot concaténé avec son miroir, c'est-à-dire un palindrome pair.*



2. *$L = \{a^n b^m, 0 \leq n \leq m \leq 2n\}$. Prenons un mot $a^n b^m$ de L. L'idée est de commencer par recopier a^n dans la pile. On obtient alors la pile $\gamma_0 A^n$. Puis on reconnaît les nb suivants, en changeant les A en B. On obtient alors la pile $\gamma_0 B^n$. On reconnaît alors les $m - nb$ restants. La pile devient $\gamma_0 B^{2n-m}$. Il n'y a plus alors qu'à vider la pile et conclure. L'automate ci-dessous reconnaît L par pile vide.*



5.2 Automates à pile et langages algébriques

Proposition 4. *Tout langage algébrique peut être reconnu par un automate à pile par pile vide.*

Démonstration. Soit $G = (\Sigma, V, S, P)$ qui reconnaît L, G en forme normale de Greibach.
Soit $\mathcal{A} = (\{q_0\}, \Sigma, V, \delta, q_0, S)$, où

$$\begin{array}{rrcl} \delta : & \{q_0\} \times \Sigma \cup \{\epsilon\} \times V & \rightarrow & \mathcal{P}(\{q_0\} \times V^*) \\ & (q_0, a, A) & \mapsto & \{(q_0, \alpha) | A \rightarrow a\alpha \in P\} \\ & (q_0, \varepsilon, S) & \mapsto & \{(q_0, \varepsilon)\} \text{ si } S \rightarrow \varepsilon \in P \end{array}$$

On note $\mathcal{L}_p(\mathcal{A})$ le langage reconnu par $\mathcal{A}$ par pile vide.
Montrons que $\mathcal{L}_p(\mathcal{A}) \subseteq L$: Soit $u \in \mathcal{L}_p(\mathcal{A})$ tel que $(q_0, S) \xrightarrow{u}{}^{n} (q_0, \varepsilon)$.

Montrons par récurrence la propriété HR :

$$(q_0, A) \xrightarrow{u}{}^{n} (q_0, \varepsilon) \Rightarrow A \rightarrow_G^* u$$

$n = 1$: deux cas sont possibles. $(q_0, S) \xrightarrow{\varepsilon} (q_0, \varepsilon)$ si et seulement si $S \rightarrow \varepsilon$ et $(q_0, A) \xrightarrow{a} (q_0, \varepsilon)$ si et seulement si $A \rightarrow a \in P$. HR est donc vérifiée.

Supposons HR vraie jusqu'au rang n. $|u| \geq 1$. Soit $u = av$.

$(q_0, A) \xrightarrow{u}_{\mathcal{A}}^{n+1} (q_0, \varepsilon)$, donc $(q_0, A) \xrightarrow{a}_{\mathcal{A}} (q_0, A_1 \cdots A_p) \xrightarrow{v}_{\mathcal{A}}^{n} (q_0, \varepsilon)$. Par construction, $A \rightarrow aA_1 \cdots A_p \in P$.

On définit v_1 comme le plus petit préfixe de v tel que après lecture de v_1 l'état de la pile est $A_2 \cdots A_p$. On définit $v_2, \cdots, v_p$ de la même manière. On a $\forall i \in \{1, \cdots, p\}$, $(q_0, A_i) \xrightarrow{v_i}{}^{k_i} (q_0, \varepsilon)$. Comme $k_i \leq n$, en appliquant HR on obtient :

$$A_i \rightarrow_G^* v_i$$

Donc $A \rightarrow_G aA_1 \cdots A_p \rightarrow_G^* av_1 \cdots v_p = u \in P$.

Montrons que $L \subseteq \mathcal{L}_p(\mathcal{A})$:
Soit $u \in L$.
Montrons par récurrence que :

$$A \rightarrow_G^n u \Rightarrow (q_0, A) \stackrel{u}{\rightarrow}^* (q_0, \varepsilon)$$

$n = 1 : A \rightarrow u$.
On a donc soit $S \rightarrow \varepsilon$, ce qui donne $(q_0, S) \stackrel{\varepsilon}{\rightarrow}_{\mathcal{A}} (q_0, \varepsilon)$, soit $A \rightarrow a$, ce qui donne $(q_0, A) \stackrel{a}{\rightarrow}_{\mathcal{A}} (q_0, \varepsilon)$.

Supposons que c'est vrai jusqu'au rang n : $A \rightarrow_G^{n+1} u = av$
Alors $A \rightarrow aA_1 \cdots A_p \rightarrow^n av = u$
Par le lemme fondamental, $\exists v_1, \cdots, v_p, \forall i \in \{1, \cdots, p\}, A_i \rightarrow^{k_i} v_i$
Par hypothèse de récurrence, $(q_0, A_i) \stackrel{v_i}{\rightarrow}_{\mathcal{A}}^* (q_0, \varepsilon)$.
D'où le résultat puisque l'on a

$$(q_0, A) \stackrel{a}{\rightarrow} (q_0, A_1 \cdots A_p) \stackrel{v_1 \cdots v_p}{\longrightarrow}^* (q_0, \varepsilon)$$

□

Proposition 5. *Tout langage reconnu par pile vide par un automate à pile est algébrique.*

Démonstration. Soit $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \gamma_0, \delta)$. On va créer une grammaire algébrique à partir des éléments de cet automate à pile.

- Si $\forall p, q \in Q, a \in \Sigma \cup \{\varepsilon\}, z \in \Gamma, (p, \varepsilon) \in \delta(q, a, z)$, alors on crée le non terminal $< qzp >$ et la règle : $< qzp > \rightarrow a$.
- Si $(p, \gamma_1 \cdots \gamma_l) \in \delta(q, a, z)$, avec les $\gamma_i \in \Gamma, a \in \Sigma \cup \{\varepsilon\}, z \in \Gamma, p, q \in Q$, alors pour chaque $q_i \in Q, q' \in Q$, on pose

$$< qzq' > \rightarrow a < p\gamma_1 q_1 >< q_1\gamma_2 q_2 > \cdots < q_{l-1}\gamma_l q' >$$

- On ajoute pour chaque $q' \in Q$, la règle $S \rightarrow < q_0\gamma_0 q' >$. S sert d'axiome.

Montrons donc que $(q, z) \stackrel{u}{\rightarrow}^* (p, \varepsilon) \Leftrightarrow < qzp > \rightarrow_G^* u$ (HR).

$\Rightarrow$: montrons par récurrence sur n que $(q, z) \stackrel{u}{\rightarrow}^n (p, \varepsilon) \Rightarrow < qzp > \rightarrow_G^* u$.

- Si $n = 1$ alors $(p, \varepsilon) \in \delta(q, u, z)$ avec $u = \varepsilon$ ou $u = a \in \Sigma$. Puisque par construction, on a la règle $< qzp > \rightarrow u$, on a bien $< qzp > \rightarrow_G^* u$.
- Supposons HR vraie pour tout $k \leq n$. Pour $n + 1$, on a $(q, z) \stackrel{a}{\rightarrow} (q', \gamma') \stackrel{u'}{\rightarrow}^n (p, \varepsilon)$ avec $a \in \Sigma \cup \{a\}$, $(q', \gamma') \in \delta(q, a, z)$ et $\gamma' \neq \varepsilon$. Soit donc $\gamma' = \gamma_1 \cdots \gamma_l$ avec $\gamma_i \in \Gamma$. Comme $(q', \gamma_1 \cdots \gamma_l) \stackrel{u'}{\rightarrow}^* (p, \varepsilon)$, cela implique qu'il existe une décomposition de u' en $u'_1 \cdots u'_l$ telle que

$$(q', \gamma_1 \cdots \gamma_l) \stackrel{u'}{\rightarrow}^{n_1} (q'_1, \gamma_1 \cdots \gamma_{l-1})$$

De même pour l'effacement des γ_i suivants. Comme les n_i sont plus petits que n, HR implique $\forall i, < q'_{i-1}\gamma_i q'_i > \rightarrow^* u'_i$. Donc

$$< q'\gamma_1 \cdots \gamma_l > \rightarrow^*_G u'_1 \cdots u'_l = u'$$

$\Leftarrow$: montrons par récurrence sur n que $< qzp > \rightarrow^n_G u \Rightarrow (q, z) \xrightarrow{u}{}^* (p, \varepsilon)$ (HR).

- Si $n = 1$, $< qzp > \rightarrow_G u$. Donc $< qzp > \rightarrow u$ est une règle de la grammaire. Par construction, $(p, \varepsilon) \in \delta(q, u, z)$ et donc $(q, z) \xrightarrow{u} (p, \varepsilon)$.
- Supposons HR vraie pour tou t$k \leq n$. On décompose la dérivation en $< qzp > \rightarrow_G \alpha \rightarrow^n_G u$, avec $< qzp > \rightarrow \alpha$ dans G. Donc $< qzp > \rightarrow a < q'\gamma_1 q_1 >< q_1\gamma_2 q_2 > \cdots < q_{l-1}\gamma_l q_l >$. Ce qui entraîne que $u = au'$ avec $u' = u'_1 \cdots u'_l$. D'après le lemmen fondamental, $< q'\gamma_1 q_1 > \rightarrow^{n_1} u'_1$ et $\forall i, < q_i\gamma_{i+1}q_{i+1} > \rightarrow^{n_i} u'_i$. Mais à chaque pas de dérivation dans G correspond un pas de dérivation dans l'automate. On a bien une dérivation dans l'automate au total.

□

On est en mesure maintenant de prouver correctement la proposition suivante :

Proposition 6. *L'intersection d'un langage rationnel L_1 et d'un langage algébrique L_2 est algébrique.*

Démonstration. Soient $\mathcal{A}_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ et $\mathcal{A}_2 = (Q_2, \Sigma, \Gamma, \delta_2, q_{02}, \gamma_0, F_2)$ reconnaissant respectivement L_1 et L_2.
On construit l'automate produit avec :
$((q_1, q_2), z) \xrightarrow{a} ((q'_1, q'_2), \gamma_1 \cdots \gamma_p)$ si $q_1 \xrightarrow{a} q'_1$ et $(q_2, z) \xrightarrow{a} (q'_2, \gamma_1 \cdots \gamma_p)$
$((q_1, q_2), z) \xrightarrow{\varepsilon} ((q'_1, q'_2), \gamma_1 \cdots \gamma_p)$ si $(q_2, z) \xrightarrow{\varepsilon} (q'_2, \gamma_1, \cdots, \gamma_p)$ □

5.2.1 Langages déterministes

Définition 10. *Un automate à pile est déterministe si :*

$$\forall q \in Q, \forall z \in \Gamma, \begin{array}{lcl} \delta(q, \varepsilon, z) \neq \varnothing & \Rightarrow & \forall a \in \Sigma, \delta(q, a, z) = \emptyset \, et \, |\delta(q, \varepsilon, z)| = 1 \\ \delta(q, \varepsilon, z) = \varnothing & \Rightarrow & \forall a \in \Sigma, |\delta(q, a, z)| \leq 1 \end{array}$$

Le déplacement dans l'automate est entièrement déterminé par le mot à lire et la configuration de départ.

Attention, si on se limite à la classe des automates déterministes, il n'y a plus équivalence entre la reconnaissance par pile vide et la reconnaissance par état terminal. Par exemple, soit $\Sigma = \{a_1, \cdots, a_n\}$ et $\overline{\Sigma} = \{\overline{a_1}, \cdots, \overline{a_n}\}$. Considérons la grammaire $G = S \rightarrow Sa_iS\overline{a_i}S|\varepsilon$. $L(G)$ est reconnu par état final et pas par pile vide avec un automate déterministe. L'intuition de ce

phénomène peut être capturée en considérant la mise en forme normale de Greibach de la grammaire : $S \to aSA|aSAB|\varepsilon; A \to \overline{a}S; B \to aSA|aSAB$

On voit apparaître une ambiguïté qui peut être résolue en repartant d'un état terminal alors que l'atteinte de la pile vide bloque la reconnaissance.

Soit $Rec(\Sigma)$ les langages reconnus par pile vide par des automates à pile déterministes et $Det(\Sigma)$ les langages reconnus par état terminal par des automates à pile déterministes. Ce sont ces derniers que l'on appelle en général langages algébriques déterministes.

Proposition 7.

$$Rec(\Sigma) \subsetneq Det(\Sigma) \subsetneq Alg(\Sigma)$$

En particulier, $\{a^n b^m | n \leq m\} \in Det(\Sigma) \backslash Rec(\Sigma)$ *et* $\{a^n b^n | n \in \mathbb{N}\} \cup \{a^n b^{2n} | n \in \mathbb{N}\} \in Alg(\Sigma) \backslash Det(\Sigma)$.

Chapitre 6

Grammaires contextuelles et machines à mémoire linéairement bornée

6.1 Définitions et exemples

6.1.1 Définition

On rappelle la définition générale des grammaires et la forme des grammaires dites contextuelles.

Définition 1. *Une grammaire contextuelle est un quadruplet* $G = (\Sigma, V, S, P)$*, où*

- Σ *est l'alphabet des* terminaux *;*
- V *est l'alphabet des* non-terminaux *(ou des* variables*) ;*
- $S \in V$ *est l'*axiome
- $P \subseteq (\Sigma \cup V)^* V (\Sigma \cup V)^* \times (\Sigma \cup V)^*$ *est un ensemble fini de règles de productions de la forme* $\varphi_1 A \varphi_2 \to \varphi_1 \omega \varphi_2$ *avec* $\varphi_1, \varphi_2, \omega \in (\Sigma \cup V)^*$

Une règle s'interprète comme : dans le contexte que forme φ_1 à gauche et φ_2 à droite, le non-terminal A se réécrit en ω.

Les grammaires algébriques correspondent aux grammaires contextuelles avec $\varphi_1 = \varphi_2 = \varepsilon$. C'est pourquoi on les appelle aussi les grammaires hors-contexte.

6.1.2 Règles non décroissantes

Il apparaît que si on impose que ω soit non vide dans la définition, une grammaire contextuelle peut être vue comme une grammaire en fait très générale dans laquelle on demande juste que les règles $\alpha \to \beta$ soient telles que $|\alpha| \leq |\beta|$. Il suffit donc de vériier la non décroissance des règles.

Donc, en fait, toute grammaire non décroissante peut être transformée en grammaire contextuelle.

Soit $\alpha = \alpha_1 \cdots \alpha_k$ et $\beta = \beta_1 \cdots \beta_k \cdots \beta_{k+l}$. On construit les règles suivantes :

$$\begin{array}{lcl}
\alpha_1 \cdots \alpha_k & \rightarrow & \gamma_1 \alpha 2 \cdots \alpha_k \\
\gamma_1 \alpha_2 \cdots \alpha_k & \rightarrow & \gamma_1 \gamma_2 \alpha_3 \cdots \alpha_k \\
& \vdots & \\
\gamma_1 \cdots \gamma_{k-1} \alpha_k & \rightarrow & \gamma_1 \cdots \gamma_{k+l} \\
& \vdots & \\
\gamma_1 \cdots \gamma_{k+l} & \rightarrow & \beta_1 \gamma_2 \cdots \gamma_{k+l} \\
\beta_1 \gamma_2 \cdots \gamma_{k+l} & \rightarrow & \beta_1 \beta_2 \cdots \gamma_{k+l} \\
& \vdots & \\
\beta_1 \beta_2 \cdots \gamma_{k+l} & \rightarrow & \beta_1 \cdots \beta_{k+l}
\end{array}$$

Au cas où α contient des $\alpha_i \in \Sigma$, on peut introduire des non-terminaux auxiliaires A_i avec la règle $A_i \rightarrow \alpha_i$.

Il est facile de voir que toutes les règles construites sont contextuelles et que l'on a préservé le langage de la grammaire initiale.

Dans la suite de ce chapitre, on s'autorisera donc à manipuler directement des grammaires non décroissantes à la place des grammaires contextuelles.

6.1.3 Exemple

Exemple 1. *La grammaire contextuelle qui suit engendre le langage* $\{a^n b^n c^n, n > 0\}$ *:*

$$\begin{array}{l}
S \rightarrow aSBc \\
S \rightarrow aBc \\
cB \rightarrow Bc \\
aB \rightarrow ab \\
bB \rightarrow bb
\end{array}$$

La troisième règle $cB \rightarrow Bc$ *est non contextuelle et peut être remplacée par :*

$$\begin{array}{l}
C \rightarrow c \\
CB \rightarrow TB \\
TB \rightarrow TU \\
TU \rightarrow BU \\
BU \rightarrow Bc
\end{array}$$

6.2 Automate à mémoire linéairement bornée

6.2.1 Définition

Définition 2. *Un automate à mémoire linéairement bornée ("mlb" en abrégé) est un automate fini muni d'une mémoire non structurée (le ruban). Il utilise un alphabet Σ, incluant un marqueur spécial $\#$ qui sert à encadrer le mot examiné sur le ruban. Les transitions de l'automate lisent un symbole sur le ruban et éventuellement remplacent ce symbole sur le ruban et changent d'état. La tête de lecture peut ensuite éventuellement se déplacer à droite ou à gauche. Formellement, $\delta \in \Sigma \times Q \times Q \times \Sigma \times \{G, D, -\}$. Si le symbole lu est $\#$, le symbole n'est pas remplacé (on ne peut pas écrire au dela de $\#$.*

6.2.2 Langages

Initialement, on met sur le ruban un mot encadré par les symboles $\#$. L'automate commence dans son état initial et la tête de lecture est positionnée sur le marqueur de gauche. Si la tête de lecture atteint le marqueur de droite et si l'automate se trouve dans un état terminal, le mot d'entrée est accepté, sinon il est écarté.

Proposition 1. *$\mathcal{A}$ étant un automate mlb, il est possible de construire une grammaire contextuelle qui reconnaisse le même langage.*

Démonstration. Pour construire la grammaire, on prend comme terminaux l'ensemble Σ et comme non-terminaux l'ensemble $\{A_{ij}, S, T, \#\}$.. A_{ij} est associé à la situation (a_i, Q_j) dans laquelle le symbole à lire est a_i et l'automate est dans l'état Q_j. S est l'axiome. Les règles de la grammaire effectuent en sens inverse les transitions de l'automate. On considère les trois groupes de règles suivantes :

- Règles engendrant les mots $\Sigma^* A_{if}$.
  - $S \to TA_{if}$ pour toute situation (a_i, q) avec q état terminal.
  - $T \to Ta_i | a_i$ pour tout $a_i \in \Sigma$.
- Règles agissant sur les mots produits par les règles précédentes. Elles fournissent un mot terminal si et seulement si l'automate accepte ce mot. La vérification se termine en tête du mot dans la situation (a_i, q_0) ou A_{i0}.
  - Pour chaque transition $(a_i, q_j) \to (q_k, a_l, D)$, on crée une règle $a_l A_{rk} \to A_{ij} a_r$ pour tout $a_r \in \Sigma$.
  - Pour chaque transition $(a_i, q_j) \to (q_k, a_l, G)$, on crée une règle $A_{rk} a_l \to a_r A_{ij}$ pour tout $a_r \in \Sigma$.
  - Pour chaque transition $(a_i, q_j) \to (q_k, a_l, -)$, on crée une règle $A_{lk} \to A_{ij}$.
- Règle pour terminer les dérivations : $\# A_{i0} \to \# a_i$.

□

Proposition 2. *A toute grammaire, on peut associer un automate mlb acceptant le même langage.*

Démonstration. Admis. □

Cette équivalence entre la classe des langages contextuels et la classe des langages acceptés par les automates mlb (non déterministes) est connu comme le théorème de Kuroda-Landweber.

Avec les automates mlb, on récupère de la stabilité.

Proposition 3. *La classe des langages acceptés par les automates mlb déterministes est une algèbre de Boole. Connu depuis seulement 1988.*

Par contre, la plupart des problèmes de décision sont indécidables. En autres, savoir si une grammaire contextuelle engendre un langage vide, fini, infini, algébrique est indécidable.

6.3 Pourquoi linéairement borné ?

L'automate mlb peut utiliser un nombre quelconque des cases où sont inscrites les lettres du mot proposé. Ce nombre est au plus égal à la taille du mot proposé. La mémoire utilisée est cette taille à laquelle il faut rajouter une constante fixe pour compter la mémoire des états. La mémoire est donc bien une fonction linéaire de la taille du mot à analyser.

Chapitre 7

Analyse syntaxique - Grammaires LL et LR

7.1 Introduction

Un compilateur doit reconnaître si un programme est symboliquement correct ou non. On retourne dans le domaine des grammaires algébriques pour pouvoir décider.

Exemple 1. *Expressions arithmétiques*
$V = \{E, T, F\}$
$\Sigma = \{a, b, c, (,), +, *\}$
$E \to E + T | T$
$T \to T * F | F$
$F \to (E) | a | b | c$
Axiome E

Le but est le suivant : soit ω un mot de Σ^*.

Si ω n'est pas reconnu par la grammaire, l'analyseur doit fournir une indication sur l'emplacement de l'erreur.

Si ω est reconnu, l'analyseur doit fournir une dérivation, c'est-à-dire en fait l'arbre syntaxique qui structure l'expression arithmétique.

On peut conduire l'analyse de deux façons :

- descendante : on part de l'axiome et on trouve la dérivation en lisant le mot de gauche à droite, et en considérant les règles de la grammaire de gauche à droite. On note cette analyse LL. On dit qu'une analyse est $LL(k)$ si il suffit de lire les k premiers symboles pour savoir quelle règle appliquer pour obtenir une dérivation concluante.
- ascendante : on part du mot et on remonte jusqu'à l'axiome pour déterminer l'arbre de dérivation en partant des feuilles. On lit le mot toujours de gauche à droite, mais on utilise les règles de la grammaire

à l'envers en partant de la partie droite. On note LR une telle analyse. On définit de même $LR(k)$.

Exemple 2. *On transforme la grammaire précédente en :*
$E \to TE'$
$E' \to +TE'|\varepsilon$
$T \to FT'$
$T' \to *FT'|\varepsilon$
$F \to (E)|a|b|c$
Axiome E

Cette grammaire est LL(1).
Regardons par exemple $a + b * (a + c)$.
Une dérivation est :

$$E \to TE' \to FT'E \to aT'E' \to aE' \to a{+}TE' \to a{+}FT'E' \to a{+}bT'E' \to \cdots$$

Il est plus simple de construire une table d'analyse.

	E	E'	T	T'	F
a	$E \to TE'$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
b	$E \to TE'$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
c	$E \to TE'$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$+$	*erreur*	$E' \to TE'$	$\cdots$	$\cdots$	$\cdots$
$*$	*erreur*	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$($	$E \to TE'$	$E' \to \varepsilon$	$\cdots$	$\cdots$	$\cdots$
$)$	*erreur*	$\cdots$	$\cdots$	$\cdots$	$\cdots$

7.2 Analyse descendante

7.2.1 Automate LL

Analyser un mot de façon descendante va s'effectuer en exécutant la grammaire. L'idée naturelle et efficace est d'utiliser l'automate à pile correspondant. Celui-ci utilise la pile pour stocker les terminaux et non-terminaux à traiter. Traiter un non-terminal revient à empiler la partie droite de la règle (expansion). Traiter un terminal revient à le lire sur le ruban d'entrée (vérification).

Il est possible aussi de programmer directement l'analyse par un ensemble de fonctions récursives. La pile dans ce cas est implicite. Enfin, notons qu'en pratique l'analyse syntaxique s'effectue en pipe-line après l'analyse lexicale qui traite de façon efficace la fabrication de symboles de plus haut-niveau lorsque ceux-ci peuvent être définis par une expression régulière (on utilise alors un simple automate d'état fini).

Définition 3. *Soit $G = (\Sigma, V, P, S)$ une grammaire algébrique. On construit l'automate à pile non-déterministe acceptant par pile vide $\mathcal{A} = (q_0, \Sigma, \Sigma \cup V, T, S, q_0)$ où les transitions de T sont :*
- *expansions :* $\{(q_0, \varepsilon, A, q_0, \overline{\alpha}) | A \to \alpha\}$
- *vérifications :* $\{(q_0, a, a, q_0, \varepsilon) | a \in \Sigma\}$

Le problème de cet automate est son non-déterminisme qui empêche une exécution efficace. On va s'intéresser dans la suite au cas où on peut réaliser une analyse déterministe en se permettant d'aller voir en avant ("look-ahead" en anglais) k lettres pour résoudre le non-déterminisme. Les grammaires permettant cela sont dites $LL(k)$. Dans la suite, on va se concentrer sur le cas le plus simple des grammaires $LL(1)$.

Exemple 3.
- $\begin{cases} S \to aAb|b \\ A \to a|bSA \end{cases}$ *est* $LL(1)$.
- $\begin{cases} S \to \varepsilon|abA \\ A \to Saa|b \end{cases}$ *est* $LL(2)$.

7.2.2 Les ensembles *Premiers* et *Suivants*

Un analyseur $LL(1)$ doit pouvoir décider, étant donnés un non-terminal A à développer et une lettre lue a, quelle règle développer.

On définit une méthode générale :
- Une règle $A \to w$ peut être développée si $a \in Premiers(w)$.
- Une règle $A \to w$ peut être développée si $w \to^* \varepsilon$ et $a \in Suivants(A)$.

La grammaire sera $LL(1)$ ssi un tel analyseur est déterministe, c'est-à-dire qu'au plus une décision est permise pour tous A et a.

Définition 4.
- *Pour $\alpha \in (\Sigma \cup V)^*$, $a \in Premiers(\alpha)$ ssi un mot dérivé de α peut commencer par le symbole terminal a :*

$$Premiers(\alpha) = \{a \in \Sigma | \exists \beta \in \Sigma \cup V, \alpha \to^* a\beta\}$$

- *$a \in Suivants(A)$ ssi un mot dérivé de A peut être suivi du symbole a :*

$$Suivants(A) = \{a \in \Sigma | \exists \alpha, \beta \in \Sigma \cup V, S \to^* \alpha A a \beta\}$$

Ces ensembles sont calculables en temps polynomial à l'aide d'un algorithme itératif. On peut démontrer que la famille des ensembles *Premiers* est la plus petite solution du système d'inéquations ensemblistes suivant :

$Premiers(a)$	$\supseteq$	$\{a\}$
$Premiers(A)$	$\supseteq$	$Premiers(\alpha)$ si $A \to \alpha \in P$
$Premiers(\varepsilon)$	$\supseteq$	$\emptyset$
$Premiers(\alpha\beta)$	$\supseteq$	$Premiers(\alpha)$
$Premiers(\alpha\beta)$	$\supseteq$	$Premiers(\beta)$ si $Vide(\alpha)$

Le prédicat $Vide(\alpha)$ est vrai ssi $\alpha \to^* \varepsilon$.

De même, la famille des ensembles $Vide$ est la plus petite solution du système d'inéquations booléennes suivant :

$Vide(a) \geq$ faux
$Vide(A) \geq Vide(\alpha)$ si $A \to \alpha \in P$
$Vide(\varepsilon) \geq$ vrai
$Vide(\alpha\beta) \geq Vide(\alpha) \wedge Vide(\beta)$

Enfin, la famille des ensembles $Suivants$ est la plus petite solution du système d'inéquations suivant :

$Suivants(B) \supseteq Premiers(\beta)$ si $A \to \alpha B \beta \in P$
$Suivants(B) \supseteq Suivants(A)$ si $A \to \alpha B \beta \in P$ et $Vide(\beta)$

Ces plus petites solutions se calculent par approximations successives, que l'on fait croître jusqu'à atteindre un point fixe. L'existence et l'unicité de ce point fixe sont garanties par le fait que ces systèmes sont monotones (ils sont sous la forme $X \supseteq F(X)$, où F est une fonction monotone sur un treillis (théorème de Tarski).

Proposition 4. *Les langages rationnels sont $LL(1)$ (on peut les décrire par une grammaire $LL(1)$).*

Il existe des grammaires qui ne sont pas $LL(k)$ pour aucun k. C'est en particulier vrai dès qu'une règle est récursive à gauche (ce n'est pas une condition nécessaire).

Exemple 4. *La grammaire* $\begin{cases} S \to A|B \\ A \to aAb|0 \\ B \to aBbb|1 \end{cases}$ *qui décrit le langage* $\{a^n 0 b^n\} \cup \{a^n 1 b^{2n}\}$*, n'est pas $LL(k)$.*

Exemple 5. *La grammaire $S \to b|Sa$ qui décrit le langage $\{ba^n, n \geq 1\}$, n'est pas $LL(k)$.*

7.2.3 Problèmes de décision

Proposition 5. *Etant donnés une grammaire algébrique G et un entier k, on peut décider si G est $LL(k)$.*

En particulier, on peut caractériser les grammaires $LL(1)$.

Proposition 6. *Une grammaire G est $LL(1)$ ssi pour toutes les règles $A \to \alpha$ et $A \to \beta$, avec $\alpha \neq \beta$, on a :*

$$Premiers(\alpha.Suivants(A)) \cap Premiers(\beta.Suivants(A)) = \emptyset$$

Proposition 7. *Etant données deux grammaires $LL(k)$, on peut décider si elles engendrent le même langage.*

Proposition 8. *Etant donnée une grammaire G, on ne peut pas décider si il existe un entier k tel que G soit $LL(k)$.*

Proposition 9. *Etant donnée une grammaire G, on ne peut pas décider s'il existe une grammaire équivalente qui soit $LL(1)$.*

C'est cette dernière proposition qui fait que la transformation en $LL(1)$ reste du domaine des heuristiques. Par exemple, la transformation de l'exemple 1 en l'exemple 2 a utilisé une heuristique pour éliminer la récursivité gauche. Dans cet exemple 2, on a $Premiers(*FT') = \{*\}$ et $Suivants(T') = \{), +\}$, qui montre bien le caractère $LL(1)$ de la grammaire.

7.3 Analyse ascendante ($LR(k)$)

L'approche ascendante va utiliser un automate à pile (a priori non déterministe). Rappelons que k est le nombre de lettres que l'on s'autorise à regarder en avant pour pouvoir décider de la règle à utiliser. Dans la suite, on va traiter le cas $LR(0)$ et $LR(1)$.

7.3.1 $LR(0)$

Considérons l'automate d'état fini ayant pour ensemble d'états $Q_1 \cup Q_2$ avec :

- $Q_1 = \{\bullet A | A \in V\}$. Les états de Q_1 sont interprétés comme le fait que l'on s'apprête à reconnaître un dérivé de A.
- $Q_2 = \{A \to u \bullet v | A \to \alpha \in P, \alpha = uv\}$. Les états de Q_2 sont interprétés comme le fait que l'on a reconnu un mot dérivé de u et qu'il reste à reconnaître un mot dérivé de v pour pouvoir affirmer avoir reconnu un mot dérivé de A.

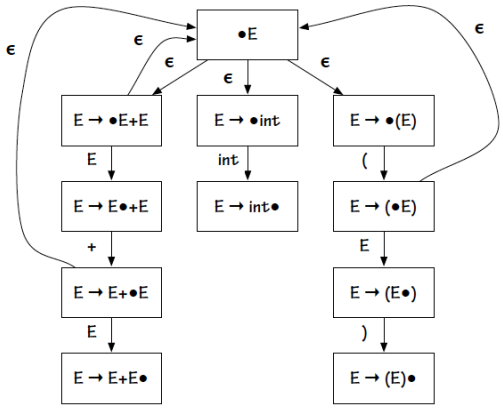
Les transitions sont étiquetées par les symboles terminaux et non terminaux ou bien par ϵ. Le jeton ($\bullet$) passe une règle par une transition ε et passe un symbole par une transition étiquetée par ce symbole.

Exemple 6. *Prenons la grammaire $E \to E + E|(E)|int$. Cette grammaire étant ambiguë, on s'attend bien sûr à ce que l'automate construit soit non déterministe (on repèrera ce que l'on appelle les situations de conflit). L'automate correspondant est donnée dans la figure 6.*

L'automate utilise la pile de la façon suivante :

- Décaler : le sommet de pile est l'état s. Si le symbole courant sous la tête de lecture est un terminal a, ce symbole est lu et on empile l'état s' accessible à partir de s par un chemin $\varepsilon^* a$.
- Réduire : le sommet de pile est l'état s, étiqueté par $A \to w\bullet$, l'automate doit dépiler $|w|$ états. Ce qui découvre un état antérieur s_0. On empile alors l'état s' accessible à partir de s_0 par un chemin $\varepsilon^* A$.

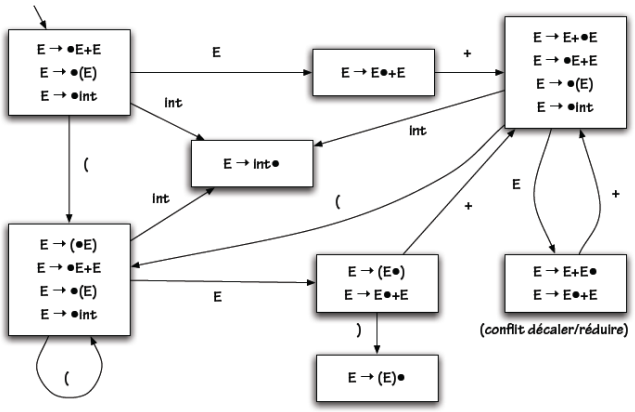
Exemple 7. *Reprenons notre exemple et considérons le mot $int + int + int$. Voici la suite des actions et l'évolution de la pile :*



Entree	*Action*	*Pile*
$int + int + int$		$(\bullet E)$
$+int + int$	*decaler*	$(\bullet E)(E \rightarrow int\bullet)$
$+int + int$	*reduire*	$(\bullet E)(E \rightarrow E \bullet +E)$
$int + int$	*decaler*	$(\bullet E)(E \rightarrow E \bullet +E)(E \rightarrow E + \bullet E)$
$+int$	*decaler*	$(\bullet E)(E \rightarrow E \bullet +E)(E \rightarrow E + \bullet E)(E \rightarrow int\bullet)$
$+int$	*reduire*	$(E \rightarrow E \bullet +E)(E \rightarrow E + \bullet E)$
	$\vdots$	

A ce point dans l'exemple, l'automate peut continuer de deux façons différentes : il peut réduire ou décaler.

L'idée est de tenter la déterminisation de l'automate par la méthode classique pour résoudre le problème. On obtient l'automate suivant.



L'automate sans ϵ peut présenter encore deux sortes de conflits :
- décaler/réduire : dans son état courant s, l'automate hésite entre interpréter ce qui a déjà été lu, ou lire un nouveau symbole.
- réduire/réduire : dans son état courant s, l'automate hésite entre deux interprétations de ce qui a déjà été lu.

Les conflits décaler/décaler n'existent plus puisque cela a été justement résolu par la détermination de l'automate de contrôle.

Pour supprimer certains conflits, on peut utiliser une construction un peu plus complexe en se donnant la possibilité de regarder en avant un symbole qui servira de prévision. C'est la technique $LR(1)$.

7.3.2 $LR(1)$

On va étendre les états par la connaissance de ce caractère d'avance, en considérant les états de la forme $A \rightarrow u \bullet v[int]$. Ces états ont la signification suivante : on a reconnu un mot dérivé de u, il reste à reconnaître le mot dérivé de v et à vérifier que le caractère suivant est int pour pouvoir affirmer avoir reconnu un mot dérivé de A. Evidemment, l'automate $LR(1)$ utilise plus de mémoire et donc a bien plus d'états que l'automate $LR(0)$.

Un conflit décaler/réduire ou réduire/réduire ne peut donc se produire que si deux actions sont possibles pour un même s et un même a.

La grammaire appartient à la classe $LR(1)$ si l'automate obtenu ne présente pas de conflit.

Proposition 10. *La classe $LR(1)$ contient strictement la classe $LL(1)$.*

La construction $LR(1)$ est coûteuse en pratique, d'où l'existence d'approximations : $SLR(1)$ et $LALR(1)$. Ces approximations engendrent des types de conflits un peu difficiles à expliquer, mais sont souvent utilisées dans les générateurs automatiques d'analyseurs syntaxiques disponibles sur étagère.

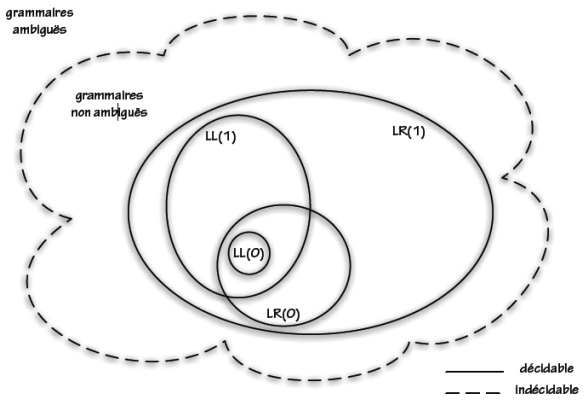
Exemple 8. *Revenons à la grammaire simple de l'exemple 6 précédent. Cette grammaire est ambiguë, donc ni $LR(0)$, ni $LL(1)$. On a un conflit décaler/réduire après avoir reconnu $E+E$ et lorsque le caractère de prévision est "+". Dans ce cas, l'expression a+a+a par exemple peut être décomposée de plusieurs façons différentes.*

Pour résoudre ce conflit, on peut essayer de réécrire la grammaire comme on l'a fait pour le cas $LL(1)$. Sans modifier la grammaire, on peut aussi indiquer explicitement à l'automate par une annotation supplémentaire si il doit réduire ou décaler. Sur l'exemple, cela reviendra à indiquer si l'opérateur "+" est associatif à gauche ou à droite. Cette seconde solution sort du cadre strict des grammaires algébriques.

En résumé, pour situer l'approche LR par rapport à l'approche LL, on peut dire qu'elle est puissante dans le fait qu'elle ne nécessite pas forcément la

transformation de la grammaire initiale. Elle demande par contre une étude fine des conflits et de leur résolution.

La figure suivante résume la hiérarchie des grammaires dans leur capacité à terme d'analyse syntaxique.



Chapitre 8

Automates d'arbres

8.1 Définitions

Considérons $\mathbb{N}$ l'ensemble des entiers positifs. On peut former des mots sur l'alphabet $\mathbb{N}$. Un alphabet ordonné est un couple (Σ, A) où Σ est un alphabet fini et A est la fonction donnant l'arité de chaque symbole de Σ : $A : \Sigma \to \mathbb{N}$. On fait l'hypothèse que Σ contient au moins un symbole d'arité 0 (les constantes). On utilisera la notation $f(, \cdots,)$ pour désigner un symbole d'arité donnée. On note $\Sigma_n \subseteq \Sigma$ les symboles d'arité n.

On considère un autre alphabet χ formé uniquement de symboles d'arité 0 (et disjoint de Σ_0) et que l'on appellera les variables.

Définition 1. *L'ensemble $T(\Sigma, \chi)$ est l'ensemble des termes sur l'alphabet ordonné Σ et l'ensemble des variables χ défini comme le plus petit ensemble vérifiant :*

- $\Sigma_0 \subseteq T(\Sigma, \chi)$,
- $\chi \subseteq T(\Sigma, \chi)$,
- *si $p \geq 1$, $f \in \Sigma_p$ et $t_1, \cdots t_p \in T(\Sigma, \chi)$, alors $f(t_1, \cdots, t_p) \in T(\Sigma, \chi)$.*

L'ensemble $T(\Sigma, \emptyset)$ définit les termes fermés (sans variables). Un terme est dit linéaire si chaque variable n'apparaît qu'une fois au plus.

Exemple 1. *Soit $\Sigma = \{cons(,), nil, a\}$ et $\chi = \{x, y\}$. Le terme $cons(x, y)$ est linéaire. Le terme $cons(x, cons(x, nil))$ n'est pas linéaire. Le terme $cons(a, cons(a, nil))$ est fermé.*

Un terme $t \in T(\Sigma, \chi)$ peut être vu comme un arbre fini ordonné. Les feuilles sont des variables ou des constantes. Les nœuds internes sont les symboles d'arité strictement positives.

8.2 Automates et langages

Un mot peut être vu comme un terme unaire. Par exemple, un mot abb sur l'alphabet $\{a, b\}$ peut être vu comme le terme $t = a(b(b(\#)))$ sur l'alphabet

ordonné $\Sigma = \{a(), b(), \#\}$ où # est une constante nouvelle.

8.2.1 Automate d'arbres

Un automate d'arbre est un ensemble de règles permettant d'introduire des nœuds "états" dans un arbre et de les faire remonter vers la racine. On parlera de reconnaissance ascendante (puisque les informaticiens représentent les arbres avec la racine en l'air...).

Définition 2. *Un automate d'arbres fini non déterministe (NFTA) sur un alphabet Σ est le n-uplet $\mathcal{A} = (Q, \Sigma, F, \delta)$, où Q est un ensemble fini d'états d'arité un, $F \subseteq Q$ l'ensemble des états terminaux. δ est l'ensemble des transitions, données sous la forme :*

$$f(q_1(x_1), \cdots, q_n(x_n)) \to q(f(x_1, \cdots, x_n))$$

où $n \geq 0$, $f \in \Sigma_n$, $q, q_1, \cdots, q_n \in Q$, $x_1, \cdots, x_n \in \chi$.

Un tel automate travaille sur un terme fermé de Σ. Il n'y a pas vraiment d'état initial dans un NFTA, mais on considère les règles $a \to q(a)$ qui introduisent les états sur les feuilles.

On peut définir la notion de pas de transformation (noté $\to$) de l'arbre fermé initial en considérant la façon dont les règles transforment l'arbre. Une suite de pas est notée $\to^*$.

Exemple 2. *Prenons $\Sigma = \{f(,), g(), a\}$ et l'automate défini par $Q = \{q(), q'(), q''()\}$, $F = \{q''()\}$ et*

$$\delta = \begin{cases} a & \to & q(a) \\ g(q'(x)) & \to & q'(g(x)) \\ g(q(x)) & \to & q'(g(x)) \\ f(q'(x), q'(y)) & \to & q''(f(x, y)) \end{cases}$$

On a $f(a, a) \to f(q(a), a) \to f(q(a), q(a))$ et
$f(g(a), g(a)) \to^* f(g(q(a)), g(q(a))) \to^* f(q'(g(a)), q'(g(a))) \to q''(f(g(a), g(a)))$

Un terme fermé t est accepté par un NFTA si $t \to^* q(t)$ pour un état q terminal. On voit bien que dans le cas d'un alphabet Σ unaire, on retombe sur la notion classique d'acceptation de mots.

Le langage d'arbres reconnu par un automate d'arbres est l'ensemble des termes fermés acceptés. Deux NFTA sont équivalents si ils reconnaissent le même langage.

Les automates considérés ici sont a priori non déterministes puisque deux règles peuvent avoir la même partie gauche.

Pour simplifier, on peut écrire les règles de transitions sous la forme $f(q_1, \cdots, q_n) \to q$ avec la signification $f(q_1(x_1), \cdots, q_n(x_n)) \to q(f(x_1, \cdots, x_n))$.

Une autre pratique est de se souvenir des états qui ont été associés aux nœuds pendant la dérivation (c'est évidemment possible, car les états introduits ne peuvent pas être remis en cause).

Exemple 3. *Soit* $\Sigma = \{or(,), and(,), not(), 0, 1\}$ *et l'automate à deux états* $Q = \{Vrai(), Faux()\}$ *avec* $F = \{Vrai()\}$ *et*

$$\delta = \begin{cases} 0 \to Faux & 1 \to Vrai \\ not(Faux) \to Vrai & not(Vrai) \to Faux \\ and(Faux, Faux) \to Faux & and(Faux, Vrai) \to Faux \\ and(Vrai, Faux) \to Faux & and(Vrai, Vrai) \to Vrai \\ or(Faux, Faux) \to Faux & or(Faux, Vrai) \to Vrai \\ or(Vrai, Faux) \to Vrai & or(Vrai, Vrai) \to Vrai \end{cases}$$

Cet automate reconnait donc les formules booléennes fermées vraies (noter bien ce mélange entre syntaxe : les termes, et sémantique : le calcul de l'automate).

$and(not(or(0,1)), or(1, not(0))) \to^*$
$Faux(Faux(Vrai(Faux, Vrai)), Vrai(Vrai, Vrai(Faux)))$

8.2.2 NFTA avec ε transitions

Comme dans le cas des mots, on peut considérer des transitions spontanées qui vont s'écrire $q \to q'$ où l'état d'un nœud peut changer spontanément.

Proposition 1. *Si un langage d'arbre est reconnu par un NFTA avec des* ε *transitions, il est aussi reconnu par un NFTA sans* ε *transition.*

Démonstration. C'est le même principe que pour les mots. On calcule les états que l'on peut obtenir à partir d'un état q et d'une suite de transitions ε (la clôture de q). Comme le NFTA peut être non déterministe, on remplace la règle qui produit l'état q par un ensemble de règles produisant les états de la clôture de q. □

8.2.3 Automates déterministes (DFTA)

Pour qu'un automate soit déterministe, on demande qu'il n'existe pas deux règles avec la même partie gauche et qu'il n'y ait pas de transitions ε. On demande aussi en général que l'automate soit complet (il y a toujours une règle pour dériver un nœud de Σ) et ne contienne pas d'états non accessibles (un état est accessible si il existe un terme fermé qui peut produire cet état comme racine).

Déterminiser, compléter et réduire sont des opérations décidables

Proposition 2. *Soit un langage d'arbres L reconnaissable par un NFTA, il existe un DFTA qui accepte ce même langage.*

Démonstration. La preuve repose sur une construction de sous-ensembles similaire à ce que l'on fait pour les mots. □

Exemple 4. *Soit* $\Sigma = \{f(,), g(), a\}$, $Q = \{q, q', q''\}$, $F = \{q''\}$, *et*

$$\delta = \left\{ \begin{array}{l} a \rightarrow q \\ g(q) \rightarrow q \\ g(q) \rightarrow q' \\ g(q') \rightarrow q'' \\ f(q,q) \rightarrow q \end{array} \right.$$

L'automate déterministe équivalent est donné par $Q = \{\{q\}, \{q, q'\}, \{q, q', q''\}\}$, $F = \{\{q, q', q''\}\}$ *et*

$$\delta = \left\{ \begin{array}{l} a \rightarrow \{q\} \\ g(\{q\}) \rightarrow \{q, q'\} \\ g(\{q, q'\}) \rightarrow \{q, q', q''\} \\ g(\{q, q', q''\}) \rightarrow \{q, q', q''\} \end{array} \right\} \cup \{f(s, s') \rightarrow \{q\} | s, s' \in Q\}$$

8.3 Propriétés des langages d'arbres reconnaissables

Commençons par donner un langage d'arbres non reconnaissable : $\Sigma = \{f(,), g(), a\}$ et $L = \{f(g^i(a), g^i(a)) | i > 0\}$. En effet, supposons que L est reconnu par un automate ayant k états, et considérons le terme $t = f(g^k(a), g^k(a))$. $t \in L$, donc il existe une dérivation réussie sur t. Il existe donc deux nœuds de la première branche qui portent le même état. Coupons cette branche à la première occurrence de cet état. On obtient le terme $t' = f(g^j(a), g^k(a))$ avec $j < k$ qui est accepté, d'où la contradiction.

Pour énoncer le lemme de l'étoile, il faut introduire la notion de hauteur d'un terme et la notion de contexte.

Définition 3. *La hauteur d'un terme* t, $H(t)$ *est définie par :*
- $H(t) = 0$ *si* $t \in \chi$,
- $H(t) = 1$ *si* $t \in \Sigma_0$,
- $H(f(t_1, \cdots, t_n) = 1 + max(H(t_i), \cdots, H(t_n))$.

Etant donné un terme linéaire C, on note $C[t_1, \cdots, t_n]$ le terme obtenu en remplaçant chaque variable x_i par le terme t_i. Pour un terme fermé t, on appelle contexte les termes linéaires à partir desquels on peut obtenir t par substitution. Il s'agit en fait des arbres préfixes.

Proposition 3. *Soit* L *un langage d'arbres reconnaissable. Il existe alors une constante* $k > 0$ *telle que pour tout terme fermé* $t \in L$ *avec* $H(t) > k$, *il existe un contexte* C, *un contexte non trivial* C' *et un terme fermé* u *tel que* $t = C[C'[u]]$ *et,* $\forall n \geq 0$, $C[C'^n[u]] \in L$.

Proposition 4. *Les langages d'arbres reconnaissables sont clos par union, intersection et complémentation.*

Démonstration. Tout se passe exactement comme pour les automates d'état finis. □

On peut définir une équivalence entre états de la même façon que pour les automates de mots et on en déduit un algorithme de minimisation.

8.4 Reconnaissance descendante

Au lieu de calculer sur un arbre en remontant vers la racine, on peut aussi imaginer de calculer en descendant, un peu comme on le fait avec les grammaires algébriques. L'idée est de partir de la racine et d'un état initial de l'automate et descendre dans l'arbre simultanément sur tous les fils, niveau par niveau. La différence entre les deux modes de reconnaissance portera essentiellement sur les questions de non déterminisme.

Définition 4. *Un automate d'arbres descendant est un tuple (Q, Σ, I, δ) où Q est un ensemble fini de symboles unaires, $I \subseteq Q$ l'ensemble des états initiaux et δ un ensemble de règles de la forme*

$$q(f(x_1, \cdots, x_n)) \to f(q_1(x_1), \cdots, q_n(x_n))$$

On voit que les règles ont simplement été inversées par rapport aux NFTA. Les règles pour les constantes sont de la forme $q(a) \to a$.

Un terme t est reconnu si on arrive à le dériver : $q(t) \to^* t$.

Proposition 5. *La classe des langages acceptés par les NFTA descendants est la même que la classe des langages acceptés par les NFTA.*

Proposition 6. *Un automate descendant est déterministe si il n'y a qu'un seul état initial et qu'il n'y a pas deux règles ayant même partie gauche. Les automates descendants déterministes sont strictement moins puissants que les NFTA.*

Les automates descendants déterministes forment la classe des langages dits clôts par chemins.

8.5 Grammaires et expressions d'arbres

8.5.1 Grammaires régulières

Les grammaires d'arbres sont comme les grammaires de mots excepté que les objets de base sont des arbres.

Définition 5. *Une grammaire d'arbres est définie par $G = (S, N, \Sigma, R)$ avec S l'axiome, N les symboles non terminaux, Σ les symboles terminaux et R l'ensemble des règles de dérivation. Les règles sont de la forme $\alpha \to \beta$ où α et β sont des arbres de $T(\Sigma \cup N \cup \chi)$. α doit contenir au moins un non terminal.*

On va se limiter pour le moment aux règles de la forme $A \to \beta$ avec $A \in N$ et β un arbre de $T(\Sigma \cup N)$. On parlera de grammaire régulière dans ce cas.

Exemple 5. *Voici une grammaire définissant les listes d'entiers. L'axiome est List, les terminaux sont* $\{0, nil, s(), cons(,)\}$.

$$\begin{aligned} List &\to nil \\ List &\to cons(Nat, List) \\ Nat &\to 0 \\ Nat &\to s(Nat) \end{aligned}$$

On définit des suites de dérivations comme d'habitude : $List \to cons(Nat, List) \to cons(s(Nat), List) \to cons(s(Nat), nil) \to cons(s(0), nil)$

Proposition 7. *Un langage d'arbres est reconnaissable ssi il est engendrable par une grammaire d'arbres régulière.*

8.5.2 Expressions régulières

Dans l'exemple précédent, les ensembles d'entiers et les listes sont définis inductivement et font apparaître des régularités :

$$\begin{aligned} Nat &= \{0, s(0), s(s(0)), \cdots\} \\ List &= \{nil, cons(\_, nil), cons(\_, cons(\_, nil)), \cdots\} \end{aligned}$$

Pour capturer cela dans une expression, il faut pouvoir indiquer où faire les itérations. C'est plus compliqué que pour les mots car il faut pouvoir distinguer les positions des nœuds dans les arbres.

L'idée est d'utiliser des symboles supplémentaires d'arité 0 pour désigner les "trous". Le remplissage des trous s'effectue par l'opération de concaténation. Tout se passe donc comme si on avait un ensemble d'opérations de concaténation (une pour chaque type de trous) et donc d'étoile.

Exemple 6. *L'ensemble des listes de* 0*, c'est-à-dire* $\{nil, cons(0, nil), cons(0, cons(0, nil)), \cdots\}$ *pourra être représenté par l'expression* $(cons(0, \circ)^{*\circ}) \odot nil$.

Proposition 8. *Un langage d'arbres est reconnaissable ssi il peut être représenté par une expression régulière d'arbres.*

8.5.3 Langages algébriques et langages d'arbres réguliers

Il n'est pas surprenant de voir apparaître une relation forte entre langages algébriques et langages d'arbres puisque les dérivations d'une grammaire algébrique sont représentables par des arbres (le langage engendré étant le langage des feuilles).

Proposition 9.
- *Soit G une grammaire algébrique, l'ensemble des arbres de dérivations de $L(G)$ est un langage d'arbres régulier.*
- *Soit L un langage d'arbres régulier, l'ensemble des mots des feuilles est un langage algébrique.*
- *Il existe des langages d'arbres réguliers qui ne sont pas des ensembles de dérivations d'un langage algébrique.*

Regardons ce dernier point. Considérons la grammaire d'axiome S, de non terminaux S, G_1, G_2 et de terminaux $s(,), g(), a, b$ et avec les règles

$$\begin{aligned} S &\to s(G_1, G_2) \\ G_1 &\to g(a) \\ G_2 &\to g(b) \end{aligned}$$

$L(G)$ est réduit à l'arbre $s(g(a), g(b))$. Imaginons que cet arbre soit un arbre de dérivation d'une grammaire algébrique. Pour générer le premier nœud de l'arbre, il faut une règle $S \to GG$ et les règles $G \to a$ et $G \to b$ pour les feuilles. Mais alors l'arbre de dérivation $S(G(a), G(b))$ doit aussi être dans $L(G)$, ce qui n'est pas le cas (on ne peut plus synchroniser les branches). Pour retrouver l'équivalence avec les langages algébriques, il faut une contrainte de localité dans la réécriture.

8.6 Au delà des langages d'arbres réguliers ?

Les langages de mots algébriques sont donc en correspondance avec les langages d'arbres réguliers. Peut-on définir des langages d'arbres plus généraux que l'on pourrait mettre en relation avec des langages de mots plus généraux que les algébriques ?

Si on prend les grammaires d'arbres dans leur forme la plus générale, on obtient la classe des langages des machines de Turing. On s'intéresse donc à trouver des cas mieux décidables en restreignant la forme des règles de la grammaire, comme on l'a fait pour les mots.

8.6.1 Langages d'arbres hors-contexte

Définition 6. *Une grammaire d'arbres hors-contexte est une grammaire d'arbres $G = (S, N, \Sigma, R)$ avec des règles de la forme $X(x_1, \cdots, x_n) \to t$ où t est un arbre de $T(\Sigma \cup N \cup \{x_1, \cdots, x_n\})$. $x_1, \cdots, x_n \in \chi$. X est un non terminal d'arité n.*

Exemple 7. *Considérons la grammaire d'axiome $Prog$, de non terminaux $\{Prog, Nat, Fact()\}$, de terminaux $\{0, s, if(,), eq(,), not(), times(,), dec()\}$*

et de règles :

$$Prog \rightarrow Fact(Nat)$$
$$Nat \rightarrow 0$$
$$Nat \rightarrow s(Nat)$$
$$Fact(x) \rightarrow if(eq(x,0), s(0))$$
$$Fact(x) \rightarrow if(not(eq(x,0)), times(x, Fact(dec(x))))$$

$\chi = \{x\}$.

Dériver dans une telle grammaire pour passer d'un terme s à t de $T(\Sigma \cup N)$ s'effectue ssi il existe une règle $X(x_1, \cdots, x_n) \rightarrow \alpha$ et un contexte C tel que $s = C[X(t_1, \cdots, t_n)]$. t est alors le terme $C[\alpha]$ dans lequel les variables $x_1, \cdots, x_n$ de α ont été remplacées par $t_1, \cdots, t_n$.

Par exemple, on a :

$$Prog \rightarrow Fact(Nat) \rightarrow Fact(0) \rightarrow if(eq(0,0), s(0))$$

Les langages d'arbres que l'on obtient ainsi sont clos par union et concaténation. Comme dans le cas des mots, on peut définir des machines d'arbres à pile qui reconnaissent exactement cette classe de langages d'arbres.

8.6.2 Grammaires d'arbres entrantes ou sortantes

La dérivation dans les grammaires d'arbres hors-contexte commence à ressembler à l'évaluation d'un programme informatique. On va distinguer deux types de stratégie d'évaluation. Dans la stratégie sortante, on dérive d'abord les non terminaux les plus intérieurs : cela correspondra à une évaluation d'appel par valeur. Dans la stratégie entrante, on dérive d'abord les non terminaux extérieurs : cela correspondra à une évaluation d'appel par nom. Les langages générés par ces deux stratégies différentes peuvent être différents.

Exemple 8. *Soit la grammaire d'arbres G, d'axiome Exp, de non terminaux $\{Exp, Nat, Dup\}$, de terminaux $\{double, s, 0\}$ et de règles :*

$$Exp \rightarrow Dup(Nat)$$
$$Nat \rightarrow s(Nat)$$
$$Nat \rightarrow 0$$
$$Dup(x) \rightarrow double(x,x)$$

Les dérivations entrantes ont la forme :

$$Exp \rightarrow Dup(Nat) \rightarrow double(Nat, Nat) \rightarrow^* double(s^n(0), s^m(0))$$

.

Tandis que les dérivations sortantes auront la forme :

$$Exp \rightarrow Dup(Nat) \rightarrow^* Dup(s^n(0)) \rightarrow double(s^n(0), s^n(0))$$

On engendre donc des langages différents.

Proposition 10. *Soit G une grammaire d'arbres hors-contexte,*

$$L_{sortant}(G) \subseteq L_{entrant}(G) = L(G)$$

Les langages sortants sont clos par union, mais pas par intersection. Ils sont clos par intersection avec un langage d'arbres régulier.

Chapitre 9

Systèmes de Lindenmayer

9.1 Introduction

On va maintenant retourner aux langages de mots, mais on modifiant deux aspects des grammaires :

- on va demander le remplacement simultané des non terminaux au cours des dérivations,
- on va interpréter la reconnaissance d'un terminal par un déplacement graphique.

Le résultat va être de bousculer la hiérarchie classique de Chomsky et d'utiliser les grammaires comme outil significatif pour modéliser la croissance des objets naturels. On va considérer ici les systèmes OL (hors-contexte) et IL (contextuels) qui définissent des langages orthogonaux à la hiérarchie de Chomsky comme indiqué dans la figure 9.1.

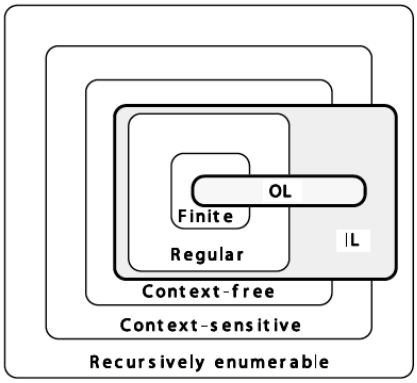


FIGURE 9.1 – Les systèmes de Lindenmayer dans la hiérarchie de Chomsky

9.2 Systèmes DOL

Commençons par regarder la classe la plus simple de L-systèmes : les DOL-systèmes (pour déterministes et hors-contexte).

Exemple 9. *On considère l'alphabet $\{a, b\}$ et des règles qui réécrivent chaque lettre en un mot. Par exemple, $a \to b$ et $b \to ab$. On part d'un mot initial et on applique toutes les substitutions définies pour chacune des lettres du mot. Puis on itère le procédé. Par exemple ici, partant du mot a, on obtient successivement les mots b, ab, bab, abbab, bababbab, etc. On peut montrer que les longueurs des mots générés est la suite de Fibonacci.*

Définition 7. *Soit un alphabet Σ. Un OL-systèmes de mots est un triplet $G = (\Sigma, w, P)$ où $w \in \Sigma^+$ est l'axiome et $P \subseteq \Sigma \times \Sigma^*$ est un ensemble fini de règles. Lorsqu'il n'existe pas de règle pour une lettre, on suppose que la lettre est recopiée simplement dans la dérivation. Le système est déterministe (DOL-système) si il y au plus une règle par lettre.*

9.3 Interprétation graphique

On considère que l'état courant du graphique (la "tortue") est donné par un triplet (x, y, α) où x, y sont les coordonnées et α la direction (de la tête).

On donne alors aux lettres $F, f, +, -$ de l'alphabet Σ la signification suivante :

- F : continuer à tracer sur une longueur d. Le nouvel état de la tortue est $(x', y', \alpha) = (x + dcos\alpha, y + dsin\alpha, \alpha)$. Une ligne est tracée du point (x, y) au point (x', y').
- f : même déplacement que F mais sans tracer le trait.
- $+$: tourner vers la gauche d'un angle δ. L'état suivant de la tortue est $(x, y, \alpha + \delta)$.
- $-$: tourner vers la droite d'un angle δ. L'état suivant de la tortue est $(x, y, \alpha - \delta)$.

Exemple 10. *Prenons l'exemple de la courbe de Koch. $\Sigma = \{F, +, -\}$, $w = F$ et $F \to F + F - F - F + F$. On prend $\delta = 90^o$. La figure 9.2 donne le résultat après 3 générations.*

9.4 Les vrais arbres

L'objectif des L-systèmes est la modélisation d'objets naturels, en particulier les plantes. Ainsi, la structure branchue des vrais arbres doit être prise en compte dans les règles de dérivation.

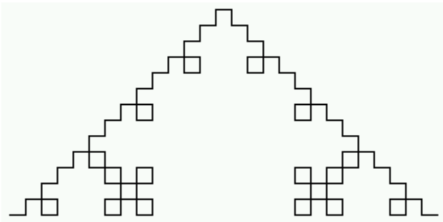


FIGURE 9.2 – La troisième génération de la courbe de Koch : $F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F+F+F-F-F+F+F+F-F-F+F-F+F-F-F+F-F+F-F-F+F+F+F-F-F+F$

9.4.1 Arbres axiaux

Un arbre axial est un type spécial d'arbre raciné. A chacun de ses nœuds de branchement, on distingue une branche principale au plus. Les autres branchements sont qualifiés de latéraux. Une suite de segments est un axe si :

- le premier segment part de la racine ou est un segment latéral partant d'un nœud,
- les segments à suivre sont principaux,
- le dernier segment n'est pas suivant par un segment principal.

Un axe avec tous ses descendants forme ce que l'on appelle une branche. Une branche est elle-même un sous-arbre axial. On ordonne les branches et les axes. L'axe partant de la racine est d'ordre 0. Un axe partant d'un segment latéral d'ordre n est d'ordre $n+1$. L'ordre d'une branche est égal à l'ordre de son axe de plus petit ordre.

9.4.2 OL-systèmes d'arbre

On rajoute à l'alphabet Σ les symboles [et] pour délimiter les branches et on utilise une pile pour gérer l'aspect emboîté des branches. L'interprétation par la tortue est la suivante :

- [: empile l'état courant de la tortue (on peut rajouter dans l'état la couleur ou l'épaisseur des traits).
-] : dépile un état. Cet état devient l'état courant de la tortue. Aucun trait n'est tracé.

Exemple 11. *Prenons l'exemple suivant :* $w = F$ *et* $F \rightarrow F[+F][-F[-F]F]F[+F][-F]$. *On prend* $\delta = 30^o$. *La figure 9.4 donne le résultat après 2 générations.*

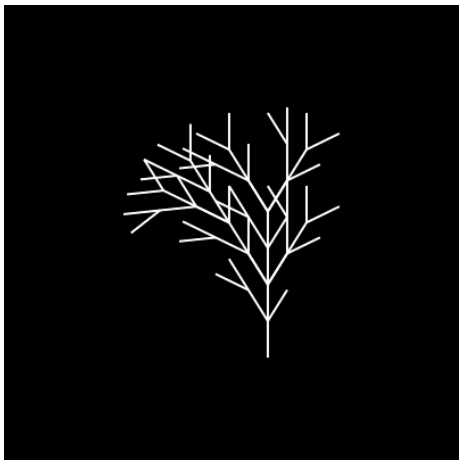


FIGURE 9.3 – La deuxième génération de la croissance d'un arbre.

9.5 L-systèmes contextuels

Comme dans les grammaires contextuelles, l'idée est que la réécriture d'une lettre est conditionnée par un contexte droit et gauche avec des règles de la forme : $a_l < a > a_r \rightarrow w$. On parle alors de $(k, l) - systme$ où le contexte gauche est un mot de longueur k et le contexte droit un mot de longueur l.

Exemple 12. *Considérons* $w = baaaaaaaaa$ *et les règles* $b < a > \rightarrow b$ *et* $< b > \rightarrow a$. *Les dérivations successives propagent le signal* b *sur la suite de* a.

L'introduction des contextes dans les systèmes d'arbres est un peu plus complexe puisque le voisinage des segments n'est pas préservé dans la dérivation. L'idée est que les suites de symboles représentant des branches ou des portions de branches peuvent être ignorées dans la recherche du contexte dans la chaîne courante. Par exemple, la règle $BC < S > G[H]M \rightarrow$ peut être utilisée pour réécrire S dans la chaîne $ABC[DE][SG[HI[JK]L]MNO]$ parce que l'on ignore la branche $[DE]$ dans le contexte gauche et la partie de branche $I[JK]L$ dans le contexte droit. Dans les outils, on peut aussi préciser les symboles à ignorer lors de la recherche du contexte.

Exemple 13.

n=26, δ=22.5°
#ignore: +-F
F1F1F1
0 < 0 > 0 → 0
0 < 0 > 1 → 1[-F1F1]
0 < 1 > 0 → 1
0 < 1 > 1 → 1
1 < 0 > 0 → 0
1 < 0 > 1 → 1F1
1 < 1 > 0 → 1
1 < 1 > 1 → 0
* < + > * → -
* < - > * → +

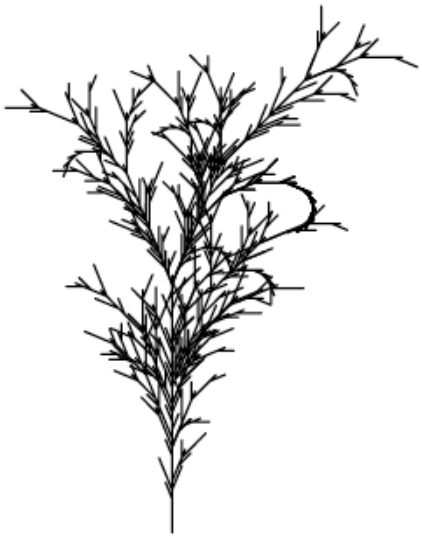


FIGURE 9.4 – Une croissance contextuelle.