AUTOMATES A PILE ET GRAMMAIRES ALGEBRIQUES

Marie-Paule Muller

Version du 31 mai 2007

Ce document est une suite au cours LANGAGES-GRAMMAIRES-AUTOMATES, que le lecteur trouvera aussi sur le site *LesMathématiques.net* . Il présente quelques méthodes mathématiques pour l'informatique théorique qui pourront servir d'entrée en matière avant l'étude d'un cours spécialisé de compilation.

L'objectif, ici, est de comprendre ce qu'est un automate à pile et son lien avec les grammaires algébriques. Cette étape supplémentaire permettra aussi d'aborder ultérieurement l'étude des machines de Turing.

Table des matières.

Marie-Paule MULLER

- *Adresse enseignement :*

IUT Robert-Schuman
B.P. 10315, F-67411 Illkirch Cedex
mél : Marie-Paule.Muller@urs.u-strasbg.fr

- *Adresse recherche :*

IRMA - UMR 7501 du CNRS
7 rue René Descartes F-67084 Strasbourg Cedex
mél : mpmuller@urs.u-strasbg.fr

Chap. 1 Automate à pile – Définitions et modèles

Conventions.

En accord avec la convention maintenant adoptée par la plupart des références bibliographiques, nous noterons ε la chaîne vide, ou une étiquette vide pour une transition.

Dorénavant, nous dirons simplement « automate » pour un ε–automate, c'est-à-dire dans le cas où des transitions d'étiquette vide sont autorisées. Lorsque l'étiquette d'une transition devra être non vide, nous le préciserons.

On rappelle que dans une pile, l'élément qui se trouve en haut est le dernier à avoir été empilé et le premier que l'on peut dépiler. En représentant une pile « à l'horizontale », par une chaîne de symboles, on écrit de gauche à droite les symboles tels qu'ils figurent de haut en bas dans la pile. Le premier symbole (celui de gauche) est celui du haut de la pile.

1.1 Introduction.

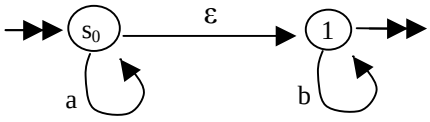
Nous avons vu que les automates reconnaissent les langages qui peuvent être engendrés par les grammaires régulières.

Le problème : peut-on décrire un dispositif plus général, associé de manière naturelle aux grammaires algébriques ?

Rappelons un exemple simple, sur l'alphabet $\Sigma = \{a, b\}$. Il est très facile d'écrire une grammaire algébrique qui engendre le langage $L = \{a^n b^n : n \geq 0\}$; il suffit pour cela de considérer la grammaire $S \rightarrow aSb \mid \varepsilon$. Néanmoins, le Lemme de l'Etoile (cf. [1]) nous permet de prouver que L ne peut pas être engendré par une grammaire régulière. L'idée intuitive est qu'avec une grammaire régulière, les symboles sont écrits consécutivement (de gauche à droite), et qu'il faut donc compter les symboles a au fur et à mesure qu'on les écrit, pour pouvoir ensuite les faire suivre d'autant de symboles b. Un dispositif qui reconnaît L doit être capable de mémoriser. En fait, il suffira de coupler un automate à une mémoire rudimentaire (une pile, supposée de capacité illimitée). Voici un tel exemple.

L'automate représenté ci-dessous est assorti d'une pile, vide au départ. En parcourant l'automate, on empile un jeton à chaque passage par la transition d'étiquette a, et on dépile un jeton – on doit pouvoir le dépiler – en passant par une transition d'étiquette b. On laisse la pile inchangée par la transition d'étiquette ε (vide) de s_0 à l'état 1. Pour qu'une chaîne testée $w \in \{a, b\}^*$ soit acceptée, on convient que la pile doit être de nouveau vide lorsque w est entièrement lue, en arrivant à l'état final. Il est facile de se convaincre que le langage reconnu est $L = \{a^n b^n : n \geq 0\}$.

L'automate sous-jacent

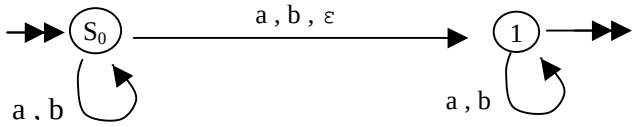


Pour d'autres langages, une pile se comportant comme un simple compteur ne suffira pas ; autrement dit, il faudra disposer de jetons marqués avec des symboles qui les différencient. Un exemple, sur l'alphabet $\Sigma = \{a, b\}$, en est le langage des palindromes.

Sur l'alphabet $\Sigma = \{a, b\}$, un *palindrome* est une chaîne qui peut se lire indifféremment de gauche à droite ou de droite à gauche. Par exemple, les chaînes baababaab , bbb , abababa , aabbaa sont des palindromes. On peut aisément écrire une grammaire algébrique qui engendre le langage des palindromes : $S \rightarrow \varepsilon \mid a \mid b \mid aSa \mid bSb$.

Il est assez facile aussi de décrire directement un automate à pile qui reconnaît ce langage : en se servant de deux symboles de pile, disons $\Gamma = \{P, Q\}$, on peut empiler ou dépiler P (resp. Q) lorsqu'on lit un symbole a (resp. b) dans la chaîne testée ; pour contraindre le dispositif à ne dépiler qu'après que tous les empilements aient été faits, il faut disposer d'une transition ad hoc (elle servira au passage du milieu d'un palindrome accepté).

En voici l'automate sous-jacent, pour lequel il restera encore à écrire précisément l'effet de chaque transition sur la pile



Mais pour tester une chaîne donnée avec un tel dispositif, autrement dit l'accepter (si c'est un palindrome) ou la rejeter (dans le cas contraire), les choses sont moins simples qu'elles en ont l'air ! Il faut non seulement « tâtonner » pour trouver le milieu de la chaîne, mais encore mémoriser (dans une pile) les symboles lus jusqu'à cet hypothétique milieu pour pouvoir les comparer ensuite, en les dépilant un à un, avec ceux qui sont lus par la suite.

1.2 Définitions et exemple.

On trouve dans la littérature plusieurs définitions, qui diffèrent légèrement. Elles correspondent en fait au choix de l'un des modes de reconnaissance possibles : par pile vide, ou par état acceptant, ou encore par pile vide *et* état acceptant. Il est donc fait mention, selon les cas, d'un symbole initial de pile et/ou d'états acceptants.

Pour certains auteurs, une transition doit nécessairement dépiler le symbole en haut de la pile avant d'empiler une chaîne de symboles ; pour d'autres, on peut ne rien dépiler (dépiler ε) avant d'empiler des symboles. Souvent, un symbole de pile initial est précisé ; il est parfois dit « de fond de pile » ou « de pile vide » (bien qu'il ne remplisse pas toujours cette fonction…).

Toutes ces variantes s'avèrent être équivalentes, dans le sens où elles reconnaissent les mêmes langages (cf. Chap. 2).

Nous adoptons le point de vue a priori le plus simple possible, avec le parti pris de n'introduire des éléments techniques supplémentaires que lorsqu'ils nous seront utiles pour la suite.

Définition 1. Un *automate à pile* est un 6-uplet $A = (\Sigma , E, s_o , F, \Gamma , \delta)$ où

- Σ est l'ensemble (fini) des symboles
- E est l'ensemble (fini) des états
- $s_o \in E$ est l'état initial (unique)
- $F \subset E$ est le sous-ensemble des états acceptants
- Γ est un ensemble fini de symboles, spécifiques à la pile
- δ est l'ensemble des transitions :

une *transition* est un triplet $([i,P] , a , [j,Q])$, où i et j sont des états,

$a \in \Sigma \cup \{\varepsilon\}$ est l'étiquette de la transition

$P, Q \in \Gamma \cup \{\varepsilon\}$.

La transition $([i,P] , a , [j,Q])$ fait passer de l'état i à l'état j en dépilant le symbole P du haut de la pile puis en empilant le symbole Q .

Pour que cette transition puisse être empruntée, il faut pouvoir dépiler P : ce symbole doit donc figurer en haut de la pile (s'il est non vide).

On peut aussi représenter une transition donnée $([i,P] , a , [j,Q])$ de manière plus visuelle, à l'aide d'une flèche munie d'une étiquette $a \in \Sigma \cup \{\varepsilon\}$:

$$[i,P] \xrightarrow{\ a\ } [j,Q]$$

Remarque. De manière un peu plus formelle, l'ensemble des transitions δ est souvent présenté comme une « relation », ou bien comme une « fonction partielle ». C'est en effet un sous-ensemble de l'ensemble $E \times (\Gamma \cup\{\varepsilon\}) \times (\Sigma \cup\{\varepsilon\}) \times E \times (\Gamma \cup\{\varepsilon\})$.

On peut aussi le voir comme une application définie sur $E \times (\Gamma \cup\{\varepsilon\}) \times (\Sigma \cup\{\varepsilon\})$, à valeurs dans l'ensemble des sous-ensembles de $E \times (\Gamma \cup\{\varepsilon\})$.

Définition 2. Pour un automate à pile $\mathcal{A}$, une chaîne $w \in \Sigma^*$ est *reconnue par état acceptant et pile vide* s'il y a une suite de transitions partant de l'état initial avec la pile vide, dont la suite des étiquettes est w , et aboutissant à un état acceptant où la pile est vide.

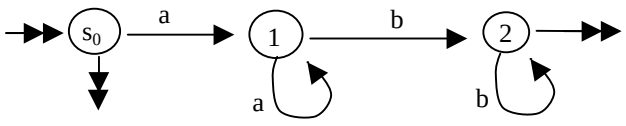
Le *langage reconnu* est l'ensemble de toutes les chaînes reconnues. Il est noté $L(\mathcal{A})$.

De manière équivalente, on dit aussi qu'une chaîne (resp. un langage) est *accepté(e)* par un automate à pile $\mathcal{A}$.

Deux automates à pile reconnaissant le même langage seront dits *équivalents*.

Nous obtenons l'automate sous-jacent en « oubliant » les symboles de pile dans les transitions. Un automate à pile peut donc se représenter comme un automate, à l'aide d'un graphe orienté et étiqueté, à condition de préciser l'action sur la pile de chacune des transitions. En fait, les symboles de pile servent d'aiguillages plutôt que de compteurs : une transition est verrouillée si le symbole à dépiler ne correspond pas au haut de pile courant.

Exemple 1. Le langage $L = \{a^n b^n : n \geq 0\}$ est reconnu par l'automate à pile à un seul symbole de pile, disons $\Gamma = \{u\}$, dont l'automate sous-jacent est



et dont les transitions sont $([s_0, \varepsilon] , a , [1, u])$

$([1 , \varepsilon] , a , [1, u])$

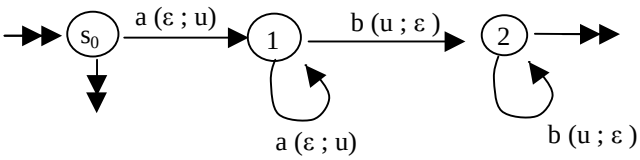
$([1 , u] , b , [2, \varepsilon])$

$([2 , u] , b , [2, \varepsilon])$

Ici, l'état initial est non récursif et il n'y a pas de transition d'étiquette vide.

Une chaîne peut être rejetée parce qu'elle est déjà refusée par l'automate sous-jacent ; elle peut aussi être rejetée du fait de la pile : on arrive à l'état final avec la pile non vide (c'est le cas pour aaabb), ou la lecture de la chaîne est bloquée parce que la pile est vide alors qu'il faudrait dépiler encore un symbole (exemple : aabbb).

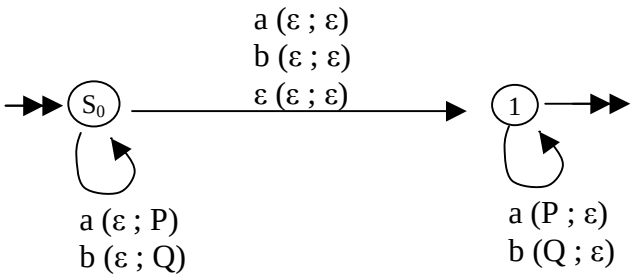
L'automate sous-jacent ne contient qu'une partie de l'information ; pour une représentation graphique complète d'un automate à pile, il faut préciser l'action sur la pile avec l'étiquette de chaque flèche. Les conventions diffèrent selon les auteurs. On peut adopter la suivante, qui a l'avantage de séparer distinctement le symbole lu dans la chaîne testée et les symboles de pile (formellement, ce sont parfois les mêmes…) :



Un automate à pile peut se décrire aussi à l'aide d'une table dans laquelle les transitions sont spécifiées. Les lignes correspondent aux états, les colonnes aux étiquettes possibles. Dans les cases figurent les triplets (état d'arrivée ; symbole dépilé ; symbole empilé) ; il peut y en avoir plusieurs par case, ou aucun. Les états particuliers sont signalés dans les dernières colonnes.

	a	b	ε	accept	
s_0	1 ; ε ; u			x	état initial
1	1 ; ε ; u	2 ; u ; ε			
2		2 ; u ; ε		x	

Exemple 2. Donnons une description complète des transitions de l'automate à pile déjà présenté pour les palindromes :



Voici la table des transitions de cet automate à pile :

	a	b	ε	accept	
s_0	s_0 ; ε ; P 1 ; ε ; ε	s_0 ; ε ; Q 1 ; ε ; ε	1 ; ε ; ε		état initial
1	1 ; P ; ε	1 ; Q ; ε		x	

1.3 Généralisation de la forme des transitions.

On généralise légèrement la notion d'automate à pile en autorisant dans les transitions l'empilement d'une chaîne de plusieurs symboles de pile.

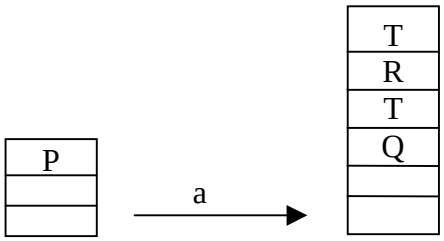
Noter qu'on ne dépile toujours qu'un symbole (ou aucun) : il faut en effet qu'un symbole soit vu en haut de la pile avant de pouvoir le dépiler, alors qu'un empilement est toujours possible.

Définition. Une transition « généralisée » d'un automate à pile est un triplet de la forme $([i,P] , a , [j, M])$, où i et j sont des états, $P \in \Gamma \cup \{\varepsilon\}$, et $M \in \Gamma^*$.

Pour fixer les idées, voici une illustration d'une telle transition. Ici, P, Q, R et T sont des symboles de pile, et la chaîne M = TRTQ est empilée après le dépilement de P :

$$[i,P] \xrightarrow{a} [j,TRTQ]$$

Par convention, c'est le symbole le plus à gauche de la chaîne M que l'on retrouvera en haut de la pile :



Proposition. Tout automate à pile ayant des transitions généralisées peut être transformé en un automate à pile qui reconnaît le même langage.

Preuve. Il suffit d'introduire suffisamment de nouveaux états pour pouvoir remplacer chaque transition « généralisée » par une chaîne de transitions convenables réalisant la même opération.

Exemple. La transition « généralisée » $[i,P] \xrightarrow{a} [j,TRTQ]$

peut être remplacée par le groupe de quatre transitions suivantes :

$$[i,P] \xrightarrow{a} [j_1 ,Q]$$

$$[j_1 , \varepsilon] \xrightarrow{\varepsilon} [j_2 ,T]$$

$$[j_2 , \varepsilon] \xrightarrow{\varepsilon} [j_3 ,R]$$

$$[j_3 , \varepsilon] \xrightarrow{\varepsilon} [j ,T]$$

les trois nouveaux états j_1 , j_2 , j_3 ayant été introduits dans ce seul but.

1.4 Restriction de la forme des transitions.

Voici une autre transformation d'un automate à pile ; elle permet de n'avoir que des transitions du modèle le plus rudimentaire possible : seule une action est effectuée à chaque étape.

De manière évidente, une transition

$$[i,P] \xrightarrow{a} [j,Q]$$

peut être remplacée par le groupe des trois transitions suivantes, en utilisant deux états supplémentaires i' et i'' que l'on ajoute à l'automate dans ce seul but :

$$[i,P] \xrightarrow{a} [i',P]$$

$$[i',P] \xrightarrow{\varepsilon} [i'', \varepsilon]$$

$$[i'', \varepsilon] \xrightarrow{\varepsilon} [j,Q]$$

Remarque 1. Nous pourrions aussi imposer la première transition ([i , ε] , a , [i', ε]) (au lieu de ([i , P] , a , [i' , P])) puisque la transition suivante doit nécessairement dépiler P . Mais dans le cas où l'automate à pile donné est déterministe (notion que nous verrons au Chap. 6), ce choix aurait l'inconvénient de ne pas conserver le déterminisme.

La proposition suivante est ainsi établie.

Proposition. Tout automate à pile peut être transformé en un automate à pile équivalent dont les transitions sont de la forme

$$[i, P] \xrightarrow{a} [j, P] \qquad a \in \Sigma \cup \{\varepsilon\}\ ,\ \ P \in \Gamma \cup \{\varepsilon\}$$

$$\text{ou} \quad [i, P] \xrightarrow{\varepsilon} [j, \varepsilon] \qquad P \in \Gamma$$

$$\text{ou} \quad [i, \varepsilon] \xrightarrow{\varepsilon} [j, P] \qquad P \in \Gamma$$

Remarque 2.

Une transition ([i , ε] , a , [j , ε]) laisse la pile inchangée quel que soit son état, même vide. Elle ne remplace la famille des transitions {([i, P] , a , [j, P]) , P ∈ Γ } que dans le cas où la pile est non vide. La transition ([i, P] , a , [j, P]) laisse aussi la pile inchangée, mais elle ne peut s'appliquer que si le symbole P figure en haut de la pile.

Plus généralement, une transition ([i , ε] , a , [j , M]) où M ∈ Γ* n'équivaut à la famille de transitions {([i , P] , a , [j , MP]) , P ∈ Γ } que si la pile est garantie non vide.

Les modifications faites jusqu'ici étaient « locales », en ce sens qu'on examinait les transitions une à une. Nous adoptons dorénavant un point de vue plus global sur les automates à pile.

1.5 Autres modes de reconnaissance. Configurations.

Nous introduisons des variantes qui ne portent pas sur la forme des transitions, mais sur l'état dans lequel doit être la pile pour qu'une chaîne soit acceptée.

On rappelle que le mode de reconnaissance « standard » que nous avons adopté (1.2 Définition 1) est celui *par état acceptant et pile vide*.

Reconnaissance par état acceptant.

Avec cette convention, une chaîne testée par un automate à pile $A = (\Sigma , E, s_o , F, \Gamma , \delta)$ est acceptée si, à partir de l'état initial, elle peut être entièrement lue en arrivant à un état de F, ceci quel que soit le contenu de la pile à ce moment-là. La pile est supposée vide au départ, mais on peut aussi convenir d'un certain contenu initial qui sera imposé.

Reconnaissance par pile vide.

Ce mode de reconnaissance autorise un automate à pile à accepter une chaîne si, à partir de l'état initial, elle peut être entièrement lue en vidant la pile. Il n'y a donc pas lieu de préciser un sous-ensemble d'états acceptants, il est sous-entendu que $F = E$ (tous les états sont acceptants).

Il y a évidemment des langages qui ne contiennent pas la chaîne vide. Afin que celle-ci ne soit pas systématiquement acceptée dès l'état initial, on convient qu'*au départ, la pile est non vide* : elle doit contenir un symbole particulier qui initialise son contenu.

Ce symbole initial de pile sera indiqué dans la définition. Ainsi par exemple, la notation $A = (\Sigma , E, s_o , \Gamma , p_1 , \delta)$, où $p_1 \in \Gamma$, désigne un automate à pile dont le contenu initial de la pile est le symbole p_1.

Cela n'implique pas que ce symbole initial de pile restera en fond de pile, ni même que la pile restera non vide durant le test d'une chaîne. Néanmoins, nous verrons comment modifier un automate à pile de manière à garantir que la pile reste non vide jusqu'à l'acceptation d'une chaîne. Ce raffinement technique sera important au moment de construire une grammaire algébrique qui produit le langage reconnu par un automate à pile donné.

Configurations.

Définition. Une *configuration* est un triplet $(i , \gamma , u) \in E \times \Gamma^* \times \Sigma^*$.

La chaîne $\gamma \in \Gamma^*$ représente le contenu (complet, lu de haut en bas) de la pile à l'état i, et $u \in \Sigma^*$ est ce qui reste encore à lire à partir de l'état i et du contenu de pile γ indiqués.

La transition $([i , P] , a , [j , M])$ permet de passer de la configuration $(i , P\eta , av)$ à la configuration $(j , M\eta , v)$, où $\eta \in \Gamma^*$, $v \in \Sigma^*$: le symbole $a \in \Sigma \cup \{\varepsilon\}$ a été consommé dans la chaîne av et le contenu de la pile est passé de $P\eta$ à $M\eta$.

Dans une *configuration initiale*, nous sommes à l'état initial s_o et la chaîne de Σ^* est une chaîne w à tester. La pile est soit vide, soit avec le symbole initial de pile s'il en est fait mention dans la spécification de l'automate à pile. Les symboles de w forment la liste des étiquettes des transitions successives qui seront à emprunter en circulant dans l'automate.

Dans une *configuration acceptante*, il n'y a plus de symboles de Σ à consommer ; elle est de la forme $(i , \gamma , \varepsilon)$, avec des conditions sur i et γ qui sont fixées par le mode d'acceptation de l'automate à pile : $i \in F$ si un ensemble F d'états acceptants est spécifié, et/ou γ vide.

La chaîne testée $w \in \Sigma^*$ est acceptée si une suite de transitions permet de passer de la configuration initiale avec w jusqu'à une configuration acceptante.

Chap. 2 Equivalence des modes de reconnaissance.

L'objectif est de prouver que si un langage est reconnu selon un certain mode de reconnaissance, alors ce langage est aussi reconnu selon n'importe quel autre de ces modes.

Les constructions qui suivent peuvent aussi être vues comme une série d'exercices permettant de s'habituer à la manipulation des automates à pile et des notions vues jusqu'à présent. Le lecteur pressé peut se reporter directement à la section 2.3 , où la forme exacte de l'automate à pile qui sera utilisé par la suite est précisée dans la Proposition.

2.1 Equivalence des modes d'initialisation (états acceptants spécifiés).

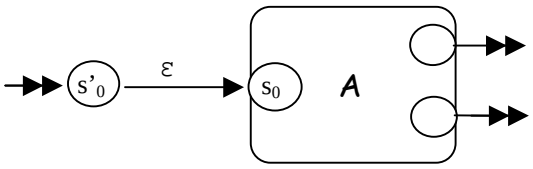
Pour les modes de reconnaissance qui spécifient des états acceptants, la question de la configuration initiale – avec ou sans symbole initial de pile – est vite réglée : ces deux types de dispositifs sont équivalents. En effet :

i) si $A = (\Sigma , E, s_o , F, \Gamma , \delta)$ est avec pile initialement vide, il suffit d'ajouter un nouvel état (non acceptant) s'_o devenant l'état initial, de choisir un symbole initial de pile p_1 dans Γ , et d'ajouter une transition $([s'_o , p_1] , \varepsilon , [s_o , \varepsilon])$ qui dépile p_1 .

L'automate $A' = (\Sigma , E \cup\{s'_o\}, s'_o , F , \Gamma , p_1 , \delta')$ ainsi obtenu reconnaît le même langage que A , avec le même mode de reconnaissance (soit par état acceptant, soit par état acceptant et pile vide).

ii) si $A = (\Sigma , E, s_o , F, \Gamma , p_1 , \delta)$ a un symbole initial de pile $p_1 \in \Gamma$, on procède de manière analogue en ajoutant une transition $([s'_o , \varepsilon] , \varepsilon , [s_o , p_1])$ qui empile p_1 . On obtient $A' = (\Sigma , E \cup\{s'_o\}, s'_o , F , \Gamma , \delta')$, avec pile initialement vide, qui reconnaît le même langage.

L'automate sous-jacent



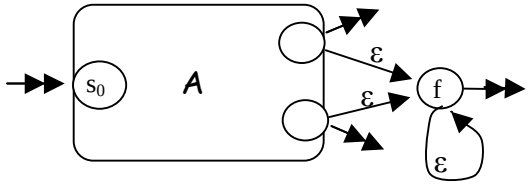
2.2 Equivalence des modes de reconnaissance.

1. Les langages reconnus par état acceptant sont reconnus par état acceptant et pile vide.

Soit $A = (\Sigma , E, s_o , F , \Gamma , \delta)$ qui reconnaît par état acceptant. Il faut arriver à vider la pile lorsqu'une chaîne est acceptée. On ajoute donc un nouvel état, disons f , acceptant, et des transitions qui vident la pile à partir des états acceptants. Leurs étiquettes sont vides et f fonctionne comme un « piège » : ainsi, le nouveau dispositif n'accepte que les chaînes acceptées par A . Précisons les transitions ajoutées :

$([i , P] , \varepsilon , [f , \varepsilon])$, pour tout $P \in \Gamma$ et pour tout $i \in F \cup\{f \}$.

L'automate sous-jacent



L'automate $A' = (\Sigma , E \cup\{f\}, s_o , F \cup\{f\}, \Gamma , \delta')$ ainsi obtenu reconnaît, par état acceptant et pile vide, le langage reconnu par A .

Cette construction peut se faire de la même façon à partir d'un automate avec symbole initial de pile $A = (\Sigma , E, s_o , F , \Gamma , p_1 , \delta)$.

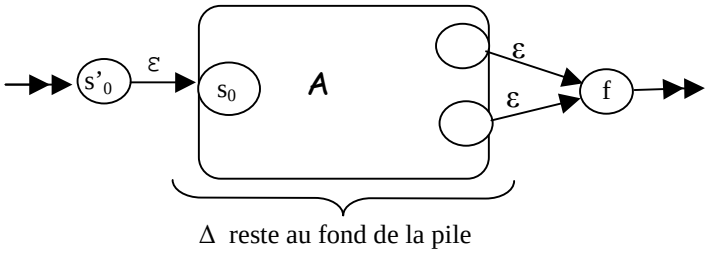
2. Les langages reconnus par état acceptant et pile vide sont reconnus par état acceptant.

Soit $A = (\Sigma , E, s_o , F , \Gamma , \delta)$ qui reconnaît par état acceptant et pile vide. Sa pile est initialement vide.

La construction d'un automate équivalent A' qui reconnaît par état acceptant est un peu plus fine que la précédente : en un état acceptant de A , il faut un moyen de reconnaître que la pile est vide et, lorsque c'est le cas, une action doit être déclenchée : transiter vers un état terminal introduit dans ce but. Il faut donc introduire aussi, via un nouvel état initial, un nouveau symbole de pile dont le seul rôle est de signaler qu'on atteint le fond de la pile (à ne pas confondre avec un simple symbole initial de pile).

On ajoute donc :

- un nouveau symbole de pile, que nous notons Δ
- deux nouveaux états : s'_o qui devient l'état initial (l'état s_o est banalisé)
  f qui sera l'unique état acceptant
- les transitions
  $([s'_o , \varepsilon] , \varepsilon , [s_o , \Delta])$ (empile Δ dans la pile initialement vide)
  $([i , \Delta] , \varepsilon , [f , \varepsilon])$ pour tout $i \in F$ (envoie vers f si Δ est détecté)



Δ reste au fond de la pile

Une fois mis dans la pile, le symbole Δ ne peut pas être dépilé par des transitions de A . Il est dépilé par les nouvelles transitions aboutissant à f . La pile n'est vide qu'aux états s'_o (initial) et f (acceptant).

L'automate à pile $A' = (\Sigma , E \cup\{s'_o , f\}, s'_o , \{f\}, \Gamma \cup\{\Delta\} , \delta')$ obtenu reconnaît, par état acceptant, le langage que reconnaît A .

Cette construction peut se faire de façon analogue à partir d'un automate qui a un symbole initial de pile $A = (\Sigma , E, s_o , F , \Gamma , p_1 , \delta)$.

Dans cette situation, le symbole Δ ajouté peut aussi être pris comme symbole initial de pile pour le nouvel automate (le symbole p_1 est banalisé).

La première transition à ajouter doit être $([s'_o , \varepsilon] , \varepsilon , [s_o , p_1])$. Elle empilera p_1 au-dessus de Δ , comme il se doit pour faire la jointure avec la « partie ancienne » du dispositif (les transitions qui sont déjà dans A).

On remarque que la pile n'est vidée qu'en f , ce qui va nous être utile dans la prochaine construction.

3. Les langages reconnus par état acceptant et pile vide sont reconnus par pile vide.

Soit A qui reconnaît par état acceptant et pile vide.

Un automate équivalent A' qui reconnaît par pile vide (avec un symbole initial de pile) s'obtient aisément en adaptant la construction ci-dessus pour laquelle, on l'a vu, la pile n'est vide qu'en f et éventuellement en s'_o (selon le mode d'initialisation de la pile).

Si $A = (\Sigma , E, s_o , F , \Gamma , \delta)$ est à pile initialement vide, il suffit d'introduire le seul état supplémentaire f en laissant s_o en état initial, mais avec Δ comme symbole initial de pile : $A' = (\Sigma , E \cup\{f \}, s_o , \Gamma \cup\{\Delta\} , \Delta , \delta')$

Dans le cas où la pile de $A = (\Sigma , E, s_o , F , \Gamma , p_1 , \delta)$ est initialisée par un symbole p_1, la construction déjà faite à la fin de *2.* convient telle quelle, il suffit de ne pas spécifier d'état acceptant : $A' = (\Sigma , E \cup\{s'_o , f \}, s'_o , \Gamma \cup\{\Delta\} , \Delta , \delta')$. La pile est maintenant initialisée par le symbole « de fond de pile » Δ .

4. Les langages reconnus par pile vide sont reconnus par état acceptant et pile vide.

Ce point est évident : tous les états sont considérés comme acceptants. On a toutefois un automate avec symbole initial de pile. Pour obtenir un automate dont la pile est initialement vide, il faut encore ajouter un nouvel état initial (non acceptant), comme en 2.1, ii).

2.3 Laquelle de ces variantes allons-nous privilégier ?

L'objectif principal à venir sera d'associer une grammaire algébrique à un langage reconnu par un automate à pile. On peut penser qu'une règle de grammaire devrait correspondre à une transition. Mais le fait de pouvoir, ou non, emprunter une transition donnée ne dépend pas seulement de l'état courant, il dépend aussi du symbole en haut de pile ; en fait, tout le parcours futur est conditionné par le contenu courant de la pile. Une variable de la grammaire devrait donc avoir pour paramètres les états et les symboles de pile. Or une règle de grammaire consiste à *réécrire* une variable : celle-ci est consommée et remplacée (par une chaîne). Par analogie, le symbole du haut de pile devra être dépilé avant d'être remplacé (par une chaîne empilée).

Ces considérations (purement intuitives) nous amèneront à privilégier les automates dont toutes les transitions commencent par dépiler un symbole de pile, donc avec une pile non vide jusqu'à l'acceptation d'une chaîne du langage (cf. Chap. 1 §1.4., Remarque 2), qui se fera par pile vide. La construction faite en 2.2. *2.* sera donc particulièrement utile.

Reprenons un automate à pile $A = (\Sigma , E, s_o , F, \Gamma , \delta)$ selon la définition standard posée (Chap.1, §1.2). La pile est initialement vide, et la reconnaissance se fait par états acceptants et pile vide.

La proposition suivante décrit la forme d'un automate à transitions généralisées (cf. Chap.1, §1.3) qui reconnaît par pile vide le même langage que A . La construction donnée dans sa preuve reprend et fait la synthèse des points *3.* et *4.* ci-dessus.

Proposition. Tout automate à pile $\mathcal{A} = (\Sigma , E, s_o , F, \Gamma , \delta)$ est équivalent à un automate à pile (à transitions généralisées) $\mathcal{A}' = (\Sigma , E', s'_o , \Gamma', \Delta , \delta')$ qui vérifie les propriétés suivantes :

- $\mathcal{A}'$ a un symbole initial de pile $\Delta \in \Gamma'$, et reconnaît par pile vide.
- Aucune transition n'aboutit à l'état initial s'_o .
- Il y a un (unique) état $f \in E'$ ayant la propriété que
  - aucune transition n'est issue de f
  - et toutes les transitions aboutissant à f sont de la forme

  $[i, \Delta] \xrightarrow{\varepsilon} [f, \varepsilon]$ *(dépilement strict de* Δ *)*
- Toutes les autres transitions ($j \neq f$) sont de l'un des types suivants, où $P \in \Gamma'$:

  $[i, P] \xrightarrow{a} [j, P]$ $\quad a \in \Sigma \cup\{\varepsilon\}$ *(la pile reste inchangée)*

  $[i, P] \xrightarrow{\varepsilon} [j, QP]$ $\quad Q \in \Gamma', \ Q \neq \Delta$ *(empilement strict de* $Q , Q \neq \Delta$ *)*

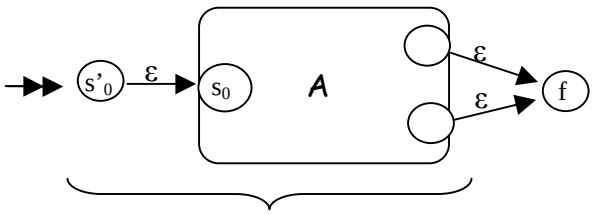
  $[i, P] \xrightarrow{\varepsilon} [j, \varepsilon]$ $\quad P \neq \Delta$ *(dépilement strict de* $P, \ P \neq \Delta$ *)*

Preuve de la proposition.

Quitte à faire les transformations « locales » décrites en 1.4 , on peut supposer que les transitions de $\mathcal{A}$ sont de la forme $([i, \varepsilon], a , [j, \varepsilon])$, $([i, P], a , [j, P])$, $([i, P], \varepsilon , [j, \varepsilon])$ ou $([i, \varepsilon], \varepsilon , [j, P])$ avec $P \in \Gamma$ et $a \in \Sigma \cup\{\varepsilon\}$. On ajoute un symbole de pile, et deux états :
$\Gamma' = \Gamma \cup \{\Delta\}$, où le symbole Δ est choisi tel que $\Delta \notin \Gamma$ (quitte à changer une notation).
$E' = E \cup \{s'_o , f \}$, où $f \notin E$ et $s'_o \notin E$. L'état s'_o est introduit dans le seul but d'obtenir la non récursivité de l'état initial.

Décrivons l'ensemble δ' des transitions :

- on admet telles quelles les transitions de $\mathcal{A}$ qui dépilent un symbole, c'est-à-dire de la forme $([i, P] , \varepsilon , [j, \varepsilon])$, avec $P \in \Gamma$
- chaque transition de $\mathcal{A}$ de la forme $([i, \varepsilon] , a , [j, Q])$ est remplacée par la famille de transitions $\{ ([i, P] , a , [j, QP]) , \ P \in \Gamma \cup\{\Delta\} \}$ (symboles a , Q vides ou non)
- on ajoute les transitions $([i, \Delta] , \varepsilon , [f , \varepsilon])$, pour tout $i \in F$
- on ajoute une transition issue de l'état initial : $([s'_o , \Delta] , \varepsilon , [s_o , \Delta])$.



L'automate sous-jacent

Δ est en fond de pile

Le symbole initial de pile Δ restera en fond de pile jusqu'à l'arrivée en f , ce qui garantit que la pile restera non vide. Le symbole en haut de la pile signale laquelle des transitions $([i, P], a , [j, QP])$ peut se substituer à $([i, \varepsilon] , a , [j, Q])$: ce symbole est dépilé, puis remis sur la pile, puis Q est empilé (cf. Chap. 1, §1.4. , Remarque 2).

Chap. 3 Automates à pile et grammaires algébriques

Les langages produits par les grammaires algébriques (dites aussi : hors-contexte) sont les langages reconnus par les automates à pile. Les constructions respectives se font explicitement.

3.1 Automate à pile associé à une grammaire algébrique.

La construction très générale qui suit paraît un peu formelle ; néanmoins, elle peut se faire directement à partir de n'importe quelle grammaire algébrique.

Théorème. Soit G une grammaire algébrique. Le langage $L(G)$ engendré par G est reconnu par un automate à pile.

Preuve du théorème.

Soit $G = (\Sigma, V, S, \boldsymbol{P})$ une grammaire algébrique. On rappelle les notations : V désigne l'ensemble des variables, l'axiome $S \in V$, et $\boldsymbol{P}$ est l'ensemble des règles de production, qui sont de la forme $A \to M$ où $A \in V$ et $M \in (\Sigma \cup V)^*$.

Pour obtenir un automate reconnaissant par pile vide (donc avec un symbole initial de pile), il nous suffit d'un état : soit $E = \{s_0\}$.

Les symboles de pile sont les symboles terminaux et les variables : $\Gamma = \Sigma \cup V$. Le symbole initial de pile est S.

Chaque règle de grammaire $A \to M$, où $M \in (\Sigma \cup V)^*$, fournit une transition d'étiquette vide, dont l'effet sur la pile est de remplacer A par la chaîne M :

$$[s_0, A] \xrightarrow{\varepsilon} [s_0, M]$$

Il reste encore à ajouter les transitions qui dépilent un symbole terminal quand il apparaît en haut de la pile. Ce symbole passe alors en étiquette. Pour tout $a \in \Sigma$:

$$[s_0, a] \xrightarrow{a} [s_0, \varepsilon]$$

L'automate à pile obtenu est de la forme $\boldsymbol{A} = (\Sigma, \{s_0\}, s_o, \Sigma \cup V, S, \delta)$.

Cette construction s'adapte aisément si l'on veut obtenir un automate à pile initialement vide qui reconnaît par état acceptant et pile vide (cf. 2.2. *4.* et 2.1. ii)).

3.2 Construction simplifiée (grammaire sous forme de Greibach).

Rappelons que toute grammaire algébrique peut être transformée, par un algorithme, en une grammaire équivalente qui est sous la *forme normale de Greibach* (voir [1]).

Nous considérons maintenant que la grammaire $G = (\Sigma, V, S, \boldsymbol{P})$ est déjà sous forme normale de Greibach. De ce fait, toutes les règles de production sont de l'un des types suivants :

- $S \to \varepsilon$ si la chaîne vide est dans le langage engendré $L(G)$.
- $A \to a\,M$ où $A \in V$, $a \in \Sigma$ et $M \in (V - \{S\})^*$

On remarquera, en particulier, que l'axiome S est non récursif.

Dans cette situation où les règles (à part la règle vide $S \to \varepsilon$) sont formées d'un symbole terminal suivi d'une chaîne de variables, la construction précédente peut être encore simplifiée : il suffit de prendre les seules variables comme symboles de pile, soit $\Gamma = V$, de mettre d'emblée en étiquette le symbole terminal qui figure en préfixe d'une règle, et d'empiler la chaîne de variables.

Pour une règle $A \to a\, A_1 A_2 \ldots A_n$: transition $[s_0, A] \xrightarrow{a} [s_0, A_1 A_2 \ldots A_n]$

Le symbole A est dépilé, la chaîne $A_1 A_2 \ldots A_n$ est empilée. On retrouve A_1 en haut de la pile, et c'est aussi la variable prête à être réécrite à la prochaine dérivation à gauche.

Si la règle vide $S \to \varepsilon$ est dans $\boldsymbol{P}$, la transition correspondante dépile simplement le symbole initial de pile S . La chaîne vide est ainsi acceptée.

L'automate à pile obtenu est de la forme $\boldsymbol{A} = (\Sigma, \{s_0\}, s_o, V, S, \delta)$.

Une suite de dérivations à gauche $S \Rightarrow w \in L(G)$ correspond à un cheminement dans l'automate à pile ; les préfixes terminaux obtenus correspondent à la suite des étiquettes lues, la pile va contenir les chaînes de variables concaténées à ces préfixes terminaux, la variable qui sera réécrite à l'étape suivante figurant en haut de la pile.

En particulier, la proposition suivante est établie.

Proposition. Soit G une grammaire algébrique. On peut construire un automate à pile reconnaissant par pile vide le langage $L(G)$, dont toutes les transitions ont une étiquette non vide, à part une transition servant exclusivement à la reconnaissance de la chaîne vide dans le cas où celle-ci est dans $L(G)$.

En testant une chaîne avec un tel automate à pile « de type Greibach », la longueur des cheminements à explorer est limitée à la longueur de la chaîne, puisque les étiquettes des transitions sont non vides.

3.3 Exemple.

Reprenons notre langage $L = \{a^n b^n : n \geq 0\}$.

Il est engendré par la grammaire algébrique G_1 : $S \to aSb \mid \varepsilon$. La première construction donne lieu aux quatre transitions suivantes, avec l'ensemble des symboles de pile $\{S, a, b\}$:

G_1 : $S \to aSb \mid \varepsilon$

$$[s_0, S] \xrightarrow{\varepsilon} [s_0, aSb]$$
$$[s_0, S] \xrightarrow{\varepsilon} [s_0, \varepsilon]$$
$$[s_0, a] \xrightarrow{a} [s_0, \varepsilon]$$
$$[s_0, b] \xrightarrow{b} [s_0, \varepsilon]$$

La grammaire G_2 suivante est équivalente à G_1 . Elle n'est pas de Greibach du fait que son axiome est encore récursif, néanmoins nous pouvons prendre comme symboles de pile les seules variables S, B :

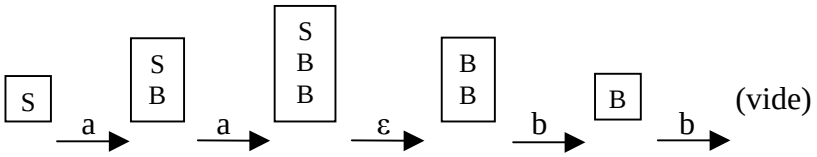
G_2 : $S \to aSB \mid \varepsilon$
$B \to b$

$$[s_0, S] \xrightarrow{a} [s_0, SB]$$
$$[s_0, S] \xrightarrow{\varepsilon} [s_0, \varepsilon]$$
$$[s_0, B] \xrightarrow{b} [s_0, \varepsilon]$$

En traduisant par un cheminement dans l'automate à pile la suite de dérivations à gauche

S => aSB => aaSBB => aaBB => aabB => aabb

le contenu de la pile sera, successivement :



La transformation en une grammaire équivalente sous forme normale de Greibach donne

G_3 : $S \rightarrow aTB \mid aB \mid \varepsilon$
$T \rightarrow aTB \mid aB$
$B \rightarrow b$

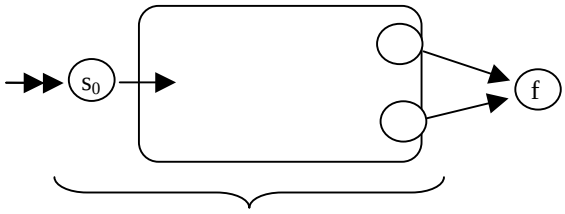
à laquelle est associé un automate à un état, symboles de pile {S, T, B }, pile initialisée par S , et six transitions faciles à expliciter. Comme l'axiome S est non récursif dans la grammaire G_3 , le symbole S ne figure en haut de pile que dans la configuration initiale.

3.4 Grammaire algébrique associée à un automate à pile.

Etant donné un langage reconnu par un automate à pile (à transitions généralisées, cf. 1.3.), notre objectif est maintenant de construire une grammaire algébrique produisant ce langage.

Les suites de dérivations à gauche à partir de l'axiome doivent être soumises aux mêmes contraintes que les cheminements partir de l'état initial dans l'automate à pile. Pour la construction d'une grammaire régulière associée à un automate (voir [1]), le paramètre d'un cheminement est l'état courant, et c'est pourquoi les états peuvent être pris comme variables de la grammaire. Dans un automate à pile, un cheminement dépend de l'état courant, mais aussi du symbole en haut de la pile. Les variables que nous devrons introduire auront donc en paramètres les états *et* les symboles de pile. Le haut de pile, prêt à être dépilé par une transition, figurera ainsi dans la première variable, prête à être réécrite lors d'une dérivation à gauche. Et le contenu de la pile se retrouvera transcrit dans une chaîne de variables.

En transformant, si nécessaire, l'automate à pile donné (cf. 1.3., 1.4. et 2.3), la construction de la grammaire peut se faire à partir d'un automate $\mathcal{A} = (\Sigma , E, s_o , \Gamma , \Delta , \delta)$ qui est du type obtenu avec la Proposition 2.3.



Δ est en fond de pile

Le point important est que la reconnaissance se fait par pile vide, et que toutes les transitions commencent par dépiler un symbole.

Nous exploiterons les propriétés supplémentaires qui contribueront à nous faire une image intuitive de la construction. Il y a un unique état où la pile peut être vidée : pour accepter une chaîne, il faut arriver en f . Par ailleurs, la forme rudimentaire des transitions servira uniquement à alléger l'écriture des règles de grammaire et des dérivations.

Passons à la construction de la grammaire associée à $\mathcal{A}$.

1. Les variables de la grammaire.

Les variables de la grammaire sont les triplets (i, P, k) où i et k sont des états, et P est un symbole de pile. La variable $S = (s_o , \Delta , f)$ joue le rôle d'axiome.

Pour une variable (i, P, k) , les paramètres i , P traduisent naturellement l'état courant et le symbole en haut de la pile. Mais quelle peut être la signification de k ? On peut interpréter l'état k comme étant un objectif « lointain », un état où est plantée une balise à atteindre ultérieurement. A l'état initial (état s_o avec Δ dans la pile), l'objectif est d'atteindre l'état f . C'est ce que dit l'axiome (s_o , Δ , f) .

Remarque. Avec cette interprétation, on voit d'ores et déjà que certaines variables seront inutiles dans la grammaire : les variables (f, P, k) parce qu'en f la pile est vide (le symbole P ne peut pas être dépilé) ; les variables (s_o , P , k) autres que (s_o , Δ , f) et les variables (i, P, s_o) parce que l'état initial s_o est non récursif.

2. Description intuitive de la grammaire.

Commençons par donner une idée intuitive de la grammaire, en adéquation avec celle que nous venons d'adopter pour les variables.

La réécriture de la variable (i, P, k) par une règle de grammaire a pour effet :

- de passer de $[i, P]$ à un état voisin j par une transition qui commence par dépiler P ,

  et/ou

- de placer une nouvelle balise, c'est-à-dire un objectif intermédiaire, en un état k' où il faudra passer avant de viser k .

Une suite de dérivations à gauche correspond, dans l'automate à pile, à un cheminement accompagné de la maintenance d'une liste de balises où l'on devra passer ultérieurement (c'est une pile, en fait, puisque la dernière balise placée est la première à être visée).

- La balise k visée par (i, P, k) peut se trouver en un état voisin, autrement dit être atteinte par une transition issue de $[i, P]$; dans ce cas (où $j = k$), ce premier objectif étant atteint, il est supprimé et on vise la balise suivante de la liste.

Pour traduire le fait que l'objectif d'un cheminement est d'arriver à l'état final, rien de plus simple : placer dès le début une balise en f . Mais comment s'assurer que tous les cheminements issus de $[s_o , \Delta]$ vont se traduire par des suites de dérivations ? En fait, on s'est arrangé pour que les contraintes imposées n'en soient pas ! Le nombre de balises sera constamment égal à la hauteur de la pile… mais cette hauteur est de toute façon inférieure ou égale au nombre de transitions qu'il faut encore emprunter pour pouvoir vider la pile. Les balises imposent des passages obligés… mais il y aura assez de variables et de règles de grammaire pour pouvoir placer *arbitrairement* les nouvelles balises, et donc tous les cheminements possibles pourront être explorés via la grammaire.

3. Les règles de grammaire.

On peut supposer que les transitions de $\mathcal{A} = (\Sigma , E, s_o , \Gamma , \Delta , \delta)$ sont de l'un des trois types décrits dans la première colonne du tableau suivant, quitte à modifier préalablement l'automate à pile (cf. Chap.2, §2.3, Proposition). Une famille de règles de grammaire est associée à chacun de ces types, pour les variables introduites en *1.*

Type de transition	**Règles de grammaire correspondantes**
Transition ne modifiant pas la pile $[i, P] \xrightarrow{a} [j, P]$ $P \in \Gamma$, $a \in \Sigma \cup \{\varepsilon\}$	pour tout k : la règle $(i , P , k) \rightarrow a (j , P, k)$ *La balise visée reste la même.*
Empilement strict d'un symbole (sur pile non vide) $[i, P] \xrightarrow{\varepsilon} [j, QP]$ $P, Q \in \Gamma$, $Q \neq \Delta$	pour tout k : les règles $(i , P , k) \rightarrow (j , Q , k')(k' , P , k)$ pour tout k' *Une nouvelle balise est placée en k' et est visée avant k .*
Dépilement strict $[i, P] \xrightarrow{\varepsilon} [j, \varepsilon]$ $P \in \Gamma$	la règle $(i , P , j) \rightarrow \varepsilon$ *La balise qui était visée en j est atteinte et supprimée.*

Remarquons que l'axiome $S = (s_o , \Delta , f)$ est non récursif, du fait l'état initial s_o est non récursif dans l'automate (dans le tableau ci-dessus, $j \neq s_o$). Et, comme prévu, les variables (f , P , k) s'avèrent inutiles puisqu'aucune transition ne part de f .

On voit aisément (par récurrence sur n) qu'une chaîne non terminale w qui dérive de l'axiome par n dérivations à gauche est de la forme suivante :

$$S \implies w = u\,(i , P_m , k_{m-1})\,(k_{m-1} , P_{m-1} , k_{m-2}) \dots (k_2 , P_2 , k_1)\,(k_1 , \Delta , f)$$

où le préfixe terminal $u \in \Sigma^*$ est suivi d'une chaîne de variables. La balise d'une variable est l'état courant pour la variable suivante. La longueur $|w| = |u|+m$ est majorée par le nombre n de réécritures faites.

4. Toute chaîne terminale produite par la grammaire est reconnue par $\mathcal{A}$.

A chaque règle de grammaire correspond une transition bien déterminée. Par récurrence sur n , on s'assure qu'à chaque suite de n dérivations à gauche

$$S \implies w = u\,(i , P_m , k_{m-1})\,(k_{m-1} , P_{m-1} , k_{m-2}) \dots (k_2 , P_2 , k_1)\,(k_1 , \Delta , f)$$

correspond, dans $\mathcal{A}$, un cheminement par n transitions, allant de la configuration initiale (état s_o , contenu de pile Δ) jusqu'à l'état i avec le contenu de pile $P_m P_{m-1} \dots P_2 \Delta$ et la chaîne des étiquettes u . La hauteur m de la pile est évidemment égale au nombre de balises placées $k_{m-1} , k_{m-2} , \dots , f$.

Considérons une chaîne complètement terminale u produite par la grammaire. Elle ne peut être obtenue que par une règle vide appliquée à une chaîne où ne figure plus qu'une seule variable :

$$S \implies \dots \implies u\,(i, \Delta , f) \implies u$$

Cette règle vide $(i, \Delta , f) \rightarrow \varepsilon$ correspond à la transition de dépilement strict de Δ . Le cheminement jusqu'à i dont l'existence est déjà garantie peut être poursuivi jusqu'à f avec cette transition qui a pour effet de vider la pile. Autrement dit, toute chaîne terminale u produite par la grammaire est reconnue par $\mathcal{A}$.

5. Toute chaîne reconnue par $\mathcal{A}$ est produite par la grammaire.

Ce point est un peu plus délicat que le précédent parce que chaque fois qu'une transition d'empilement strict est empruntée lors d'un cheminement, il faut « tâtonner » parmi toutes

les règles de grammaire qui peuvent être appliquées (paramètre k' , dans le tableau). Mais, justement, il y a suffisamment de variables et de règles pour pouvoir placer arbitrairement ces nouvelles balises.

Lemme. Etant donné un cheminement partant de s_o (pile Δ) et n'arrivant pas à f , on considère son état d'arrivée i , avec la chaîne des étiquettes u et le contenu obtenu dans la pile $P_m P_{m-1} \ldots P_2 \Delta$. La hauteur de la pile est $m \geq 1$.

Quelle que soit la liste de m-1 états $k_{m-1} k_{m-2} \ldots k_1$, il existe une suite de dérivations à gauche

$$S \Longrightarrow u (i , P_m , k_{m-1}) (k_{m-1} , P_{m-1} , k_{m-2}) \ldots (k_2 , P_2 , k_1) (k_1 , \Delta , f)$$

Application du lemme. Considérons une chaîne u acceptée, via un cheminement depuis la configuration initiale (s_o , Δ, u) jusqu'à la configuration $(f, \varepsilon, \varepsilon)$. La dernière transition empruntée vide la pile en dépilant Δ . Il s'agit donc d'une transition du type $([i , \Delta], \varepsilon , [f , \varepsilon])$, et la règle $(i , \Delta , f) \rightarrow \varepsilon$ lui est associée. Au cheminement jusqu'à i correspond, d'après le lemme, une suite de dérivations à gauche $S \Longrightarrow u (i , \Delta , f)$. Il reste à appliquer la règle vide $(i , \Delta , f) \rightarrow \varepsilon$ pour obtenir la dérivation $S \Longrightarrow u$.

Preuve du lemme. Par récurrence sur le nombre de transitions empruntées.

i) Cheminement par une seule transition.

Allant de s_o à l'état i , cette transition est du type $([s_o , \Delta] , a , [i , \Delta])$ et alors la règle $S \rightarrow a(i , \Delta , f)$ fournit la dérivation voulue, ou bien du type $([s_o , \Delta] , \varepsilon , [i , Q\Delta])$ et alors, quel que soit k_1 , la règle $S \rightarrow (i , Q , k_1)(k_1 , \Delta , f)$ fournit la dérivation voulue.

ii) Considérons un cheminement par n transitions, partant de s_o (pile Δ) et arrivant à l'état i avec le contenu de pile $P_m P_{m-1} \ldots P_2 \Delta$ et la chaîne d'étiquettes u . On le prolonge par une $(n+1)^{ème}$ transition qui mène de i à l'état j , d'étiquette a (éventuellement vide).

- Si cette dernière transition empile strictement un symbole, disons P_{m+1} , il s'agit de $([i , P_m], \varepsilon , [j , P_{m+1} P_m])$. On se donne m balises $k_m \; k_{m-1} \; \ldots \; k_1$. Avec les m-1 dernières de ces balises, par hypothèse de récurrence, on a une suite de dérivations à gauche pour le cheminement jusqu'à i :

  $$S \Longrightarrow u (i , P_m , k_{m-1}) (k_{m-1} , P_{m-1} , k_{m-2}) \ldots (k_2 , P_2 , k_1) (k_1 , \Delta , f)$$

  La dernière transition assure l'existence de la règle

  $$(i , P_m , k_{m-1}) \rightarrow (j , P_{m+1} , k_m) (k_m , P_m , k_{m-1})$$

  qu'il reste encore à appliquer pour obtenir la dérivation cherchée

  $$S \Longrightarrow u (j , P_{m+1} , k_m) (k_m , P_m , k_{m-1}) \ldots (k_2 , P_2 , k_1) (k_1 , \Delta , f)$$

- Si la transition de i à j laisse la pile inchangée, il s'agit de $([i , P_m], a , [j , P_m])$.
  Des balises $k_{m-1} \; k_{m-2} \; \ldots \; k_1$ étant données, on a une suite de dérivations à gauche

  $$S \Longrightarrow u (i , P_m , k_{m-1}) (k_{m-1} , P_{m-1} , k_{m-2}) \ldots (k_2 , P_2 , k_1) (k_1 , \Delta , f)$$

  La dernière transition assure l'existence de la règle $(i , P_m , k_{m-1}) \rightarrow a (j , P_m , k_{m-1})$
  qu'il reste à appliquer pour obtenir la suite de dérivations cherchée

  $$S \Longrightarrow ua (j , P_m , k_{m-1}) (k_{m-1} , P_{m-1} , k_{m-2}) \ldots (k_2 , P_2 , k_1) (k_1 , \Delta , f)$$

- Si la transition de i à j dépile strictement le symbole P_m , il s'agit de la transition $([i , P_m], \varepsilon , [j , \varepsilon])$. On se donne m-2 balises $k_{m-2} \; k_{m-3} \; \ldots \; k_1$. Avec ces balises plus une balise supplémentaire en j , on a une suite de dérivations à gauche

  $$S \Longrightarrow u (i , P_m , j) (j , P_{m-1} , k_{m-2}) \ldots (k_2 , P_2 , k_1) (k_1 , \Delta , f)$$

  La dernière transition assure l'existence de la règle $(i , P_m , j) \rightarrow \varepsilon$, qu'il reste à appliquer pour obtenir la dérivation cherchée (et finir la preuve par récurrence).

Chap. 4 Quelques opérations sur les langages algébriques

Il arrive fréquemment qu'un langage à étudier puisse s'exprimer à l'aide d'opérations simples appliquées à des langages algébriques connus.

4.1 Opérations régulières sur les langages algébriques.

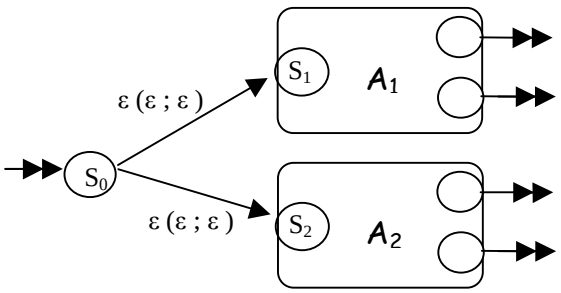
Proposition 4.1. Sur un alphabet Σ donné, les langages algébriques sont fermés pour les opérations suivantes sur les langages :
- réunion
- concaténation
- étoile de Kleene

Preuve.

Considérons deux grammaires algébriques $G_1 = (\Sigma, V_1, S_1, \boldsymbol{P}_1)$, $G_2 = (\Sigma, V_2, S_2, \boldsymbol{P}_2)$ et les langages L_1 , L_2 qu'elles engendrent.

i) On obtient une grammaire algébrique qui engendre la réunion $L_1 \cup L_2$ en réunissant tous les ingrédients et en ajoutant encore une variable supplémentaire S_0 (qui sera l'axiome) et qui se réécrit $S_0 \rightarrow S_1 \mid S_2$. Il faut juste veiller à distinguer les variables des deux grammaires (sinon, renommer certaines variables) : la réunion $V_1 \cup V_2$ doit être une réunion disjointe.

Un automate à pile reconnaissant $L_1 \cup L_2$ s'obtient en procédant de manière analogue à partir d'automates à pile respectifs. Considérons qu'ils sont à pile initialement vide, reconnaissance par état acceptant et pile vide. Il suffit alors de les réunir de façon disjointe, puis d'ajouter un nouvel état qui servira d'état initial. Noter qu'une (unique) pile suffit.

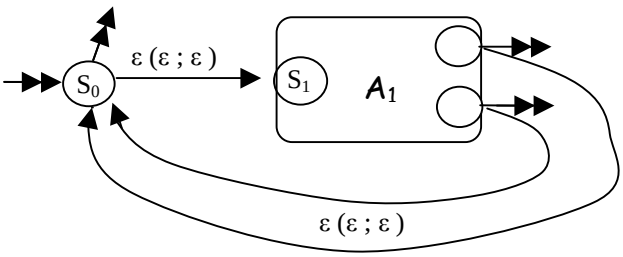


ii) Une grammaire qui engendre le langage concaténé $L_1.L_2$ se construit tout aussi facilement : réunir les deux grammaires, puis ajouter une nouvelle variable-axiome S_0 avec la règle de réécriture $S_0 \rightarrow S_1 S_2$.

Un automate à (une seule) pile s'obtient en branchant les états acceptants (dorénavant banalisés) de $\boldsymbol{A}_1$ sur l'état initial (banalisé) de $\boldsymbol{A}_2$.

iii) Pour écrire une grammaire algébrique produisant le langage L_1^* , il faut être attentif à la question de la chaîne vide qui, si elle n'est pas dans L_1 , doit néanmoins être dans L_1^* . L'éventuelle récursivité de l'axiome S_1 doit aussi être prise en compte. Une réponse générale consiste à ajouter un nouvel axiome S et les règles $S_0 \rightarrow S_1 S_0 \mid \varepsilon$.

Un automate à pile s'obtient selon le même principe



Remarque. Si l'axiome de la grammaire donnée est non récursif, on pourrait boucler directement sur lui. Mais seulement dans ce cas, comme le montre l'exemple suivant.

Notre langage $L = \{a^n b^n : n \geq 0\}$ est engendré par la grammaire $G : S \rightarrow aSb \mid \varepsilon$, dans laquelle l'axiome est récursif. En bouclant directement sur S, on obtient la grammaire $G' : S \rightarrow SS \mid aSb \mid \varepsilon$. Il est facile de voir que le langage engendré par cette grammaire G' n'est pas L^*.

En fait, $L(G')$ est le « langage des parenthèses », si l'on considère que le symbole a (resp. b) représente la parenthèse ouvrante (resp. fermante). Par exemple, les parenthèses dans l'expression $(((x+x)+x)+(x+x))$ correspondent au mot $aaabbabb$, qui est dans $L(G')$ mais non dans L^*.

4.2 Intersections et compléments de langages.

Une instant de réflexion suffit pour s'apercevoir que construire une grammaire produisant *l'intersection* de deux langages algébriques s'avère être un défi qui n'a rien de commun avec ce que nous avons fait pour une réunion. En considérant des automates à pile, le problème n'est pas plus simple : pour tester une chaîne, on en est réduit à devoir cheminer « en parallèle » dans les deux automates à pile… et donc à utiliser les deux piles à la fois. On travaille, en somme, avec un automate « à deux piles ».

Proposition. Sur un alphabet Σ donné,

i) l'intersection d'un langage algébrique et d'un langage *régulier* est un langage algébrique.

ii) les langages algébriques ne sont pas fermés pour l'opération d'intersection : on peut trouver deux langages algébriques dont l'intersection n'est pas un langage algébrique.

iii) les langages algébriques ne sont pas fermés pour la complémentation : on peut trouver un langages algébrique dont le complémentaire n'est pas un langage algébrique.

Preuve.

i) Un langage régulier est reconnu par un automate, que l'on peut même supposer déterministe et complet (cf. [1]). Un cheminement dans un automate à pile s'accompagne d'un cheminement parallèle dans cet automate : une seule pile est ainsi utilisée.

Un tel cheminement simultané dans deux dispositifs peut se formaliser de la manière suivante.

Soit $A_1 = (\Sigma , E_1 , s_1 , F_1, \Gamma , \delta_1)$ un automate à pile, et $A_2 = (\Sigma , E_2 , s_2 , F_2 , \delta_2)$ un automate. Les langages reconnus sont notés respectivement L_1 et L_2 (et L_2 est régulier). Cheminer « en même temps » dans A_1 et A_2, c'est cheminer dans un automate à pile dont l'ensemble des états est $E_1 \times E_2$. L'état initial est bien sûr (s_1 , s_2). Les symboles de pile et le mode de reconnaissance sont ceux de A_1. On associe les transitions respectives ayant même étiquette : pour $a \in \Sigma$, on admet la transition $([(i_1, i_2), P], a, [(j_1, j_2), M])$ si

et seulement si $([i_1 , P], a , [j_1 , M])$ est dans δ_1 et (i_2 , a , j_2) dans δ_2. Avec une transition d'étiquette vide dans A_1, il faut un « état stationnaire » dans A_2 : la transition $([(i_1, i_2), P], \varepsilon, [(j_1, i_2), M])$ est admise si et seulement si $([i_1 , P], \varepsilon , [j_1 , M])$ est dans δ_1.

On a tout fait pour que le langage reconnu par cet automate à pile soit $L_1 \cap L_2$.

Noter qu'il n'est pas nécessaire que l'automate A_2 soit déterministe ou complet.

ii) Sur $\Sigma = \{a, b, c\}$, voici un exemple de deux langages algébriques (ce sont des concaténations de langages algébriques connus)

$$L_1 = \{a^n b^n : n \geq 0 \}.\{c\}^* \quad \text{et} \quad L_2 = \{a\}^*.\{b^n c^n : n \geq 0 \}$$

dont l'intersection $\{a^n b^n c^n : n \geq 0 \}$ n'est pas un langage algébrique, ce qui sera prouvé au chapitre suivant, avec le Lemme de l'Etoile.

iii) On a vu que les langages algébriques sont fermés pour la réunion. S'ils étaient fermés pour la complémentation, ils devraient l'être aussi pour l'intersection, puisque pour tous langages L_1 et L_2 on a : $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ (loi de de Morgan)

Pour être plus concret, donnons un exemple explicite.

Nous avons déjà annoncé en ii) que le langage $L = \{a^n b^n c^n : n \geq 0 \}$ n'est pas algébrique. Mais son complémentaire $\{a^i b^j c^k : i, j, k \geq 0 , i \neq j \text{ ou } j \neq k\}$ est algébrique, puisqu'il est la réunion des quatre langages algébriques suivants (chacun est algébrique comme concaténation de langages algébriques connus) :

$L_{ab} = aa^*\{a^n b^n : n \geq 0 \}c^*$ $\qquad$ $L_{ba} = \{a^n b^n : n \geq 0 \}bb^*c^*$

$L_{bc} = a^*bb^*\{b^n c^n : n \geq 0 \}$ $\qquad$ $L_{cb} = a^*\{b^n c^n : n \geq 0 \}cc^*$

(on a allégé l'écriture en notant a^* au lieu de $\{a\}^*$ etc..).

Chap. 5 Le « Lemme de l'Etoile » pour les langages algébriques

Le problème : un langage L étant donné, comment savoir s'il peut être engendré par une grammaire algébrique, et donc s'il est reconnu par un automate à pile ?

Le *lemme de l'Etoile* (appelé aussi *« lemme du gonflement »* ou *« lemme de pompage »*) donne une condition que les chaînes de L doivent nécessairement satisfaire s'il y a une telle grammaire. Cette condition nécessaire sera exploitée pour prouver que certains langages ne peuvent pas être engendrés par une grammaire algébrique.

Dans [1] , un lemme de l'Etoile pour les langages réguliers a été établi à partir de la forme particulière qu'ont les arbres de dérivation dans le cas d'une grammaire régulière. Dans le cadre plus général des grammaires algébriques, l'idée est exactement la même : si un arbre de dérivation est plus haut que le nombre de variables, il y a une variable répétée sur un chemin issu de la racine S de l'arbre, et la portion de branche située entre ces deux occurrences d'une même variable peut être dupliquée autant de fois que l'on veut. Seule la mise en forme du lemme est un peu plus technique, du fait que les arbres de dérivation n'ont plus de forme particulière.

5.1 Le Lemme de l'Etoile (cas d'une grammaire algébrique).

Soit $G = (\Sigma, V, S, \boldsymbol{P})$ une grammaire algébrique, où V désigne l'ensemble des variables, $S \in V$ est l'axiome, et $\boldsymbol{P}$ est l'ensemble des règles de production.

Les règles sont de la forme $A \to M$ où $A \in V$ et $M \in (\Sigma \cup V)^*$. Dans ce qui suit, nous aurons à utiliser deux constantes liées à la grammaire : le nombre $|V|$ de variables, et la longueur maximale des règles de production $m = \max \{|M| : A \to M \in \boldsymbol{P}\}$.

On associe à toute suite de dérivations $S \Longrightarrow w \in \Sigma^*$ un arbre de dérivation. Appelons *hauteur* de l'arbre le nombre maximal d'arêtes sur les chemins descendant à partir de la racine de l'arbre. Si h est la hauteur de l'arbre, il y a au plus m^h extrémités de branches. Les symboles qui figurent aux extrémités des branches, lus de gauche à droite, forment la chaîne w . Sa longueur $|w|$ est donc certainement inférieure ou égale à m^h . Si $|w|$ est strictement supérieure à la constante $m^{|V|}$, on a donc $h > |V|$. Mais si $h > |V|$, l'arbre de dérivation contient un chemin avec deux nœuds étiquetés d'une même variable. Et ceci quel que soit l'arbre de dérivation pouvant être associé à la chaîne w .

Sur un chemin de hauteur maximale, considérons deux nœuds étiquetés par une même variable A (cf. fig. suivante). On peut en trouver dans les $|V|+1$ derniers segments..

La chaîne w s'écrit alors comme la concaténation $w = uxzyv$ de cinq chaînes que l'on peut décrire de la manière suivante : avant la première occurrence de A , les extrémités des branches qui partent vers la gauche du chemin forment la chaîne u ; à partir du premier nœud A et jusqu'à la deuxième occurrence de A , les branches à gauche du chemin forment la chaîne x ; vers le bas à partir du deuxième nœud A , on obtient la chaîne z ; à droite du chemin, les chaînes y et v sont les analogues de x et u respectivement

Ces chaînes correspondent à une suite de dérivations, à laquelle est associée le même arbre de dérivation (mais ce ne sont pas des dérivations à gauche) :

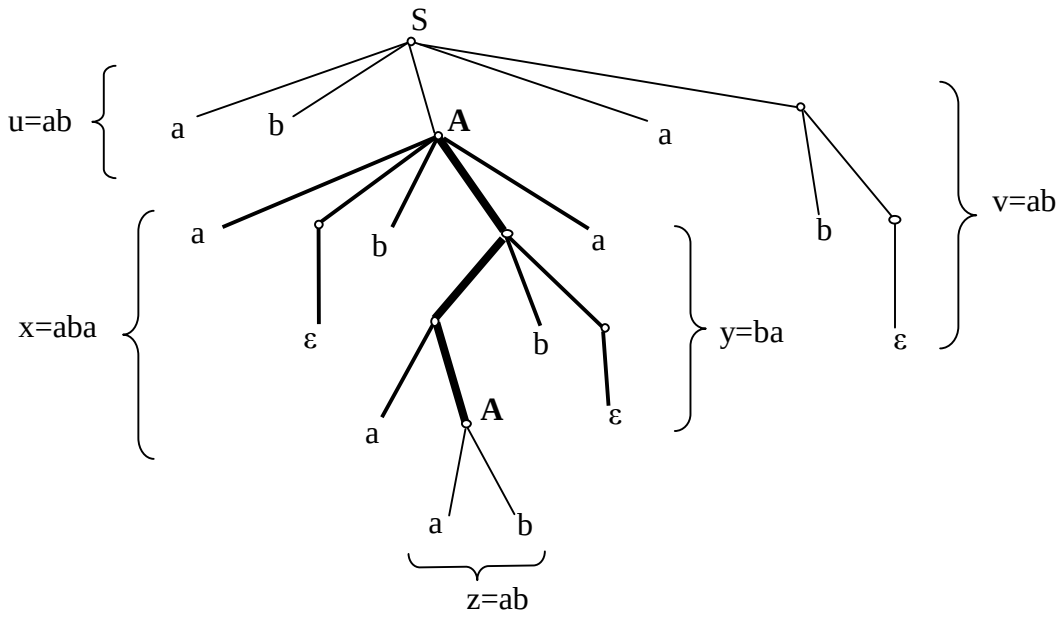
$$S \Longrightarrow uAv \Longrightarrow uxAyv \Longrightarrow uxzyv = w$$

Nous pouvons greffer des clones du tronçon d'arbre qui est compris entre les deux nœuds étiquetés par la même variable ; autrement dit, nous pouvons répéter la séquence de dérivations $A \Longrightarrow xAy$, ou la supprimer, d'ailleurs. Ceci produit les chaînes

$S \Longrightarrow uAv \Longrightarrow uzv$ et $S \Longrightarrow uAv \Longrightarrow uxAyv \Longrightarrow uxxAyyv \Longrightarrow uxxzyyv$

$S \Longrightarrow uAv \Longrightarrow uxAyv \Longrightarrow \ldots \Longrightarrow ux^nAy^nv \Longrightarrow ux^nzy^nv$

En conclusion, toutes les chaînes de la forme ux^nzy^nv avec $n \geq 0$ sont produites par la grammaire.



Il nous reste toutefois à nous assurer que nous pouvons bien obtenir des chaînes distinctes de la chaîne initiale w .

Une grammaire algébrique G quelconque peut contenir des règles vides, et de ce fait les deux chaînes x et y peuvent être vides. Mais dans ce cas, le tronçon d'arbre compris entre les deux occurrences de A peut être supprimé, le chemin est raccourci, sans que la chaîne produite soit modifiée ; on a obtenu une nouvelle suite de dérivations $S \Longrightarrow w$, associée à un arbre qui a strictement moins de nœuds, mais sa hauteur doit toujours être supérieure à $|V|$ puisque $|w| > m^{|V|}$. Cette suppression de « tronçons vides » (dérivations de la forme $A \Longrightarrow A$) peut être répétée tant que possible. On a prouvé l'existence d'un chemin porteur de nœuds étiquetés par une même variable et pour lesquels xy est non vide. De plus, on obtient une décomposition $w = uxzyv$ qui est telle que $|xzy| \leq m^{|V|+1}$ en choisissant les deux nœuds dans la partie basse du chemin (parmi ses $|V|+1$ derniers nœuds étiquetés d'une variable).

Théorème (Lemme de l'Etoile). Si un langage L peut être engendré par une grammaire algébrique, alors il existe une constante K telle que toute chaîne $w \in L$ dont la longueur est supérieure ou égale à K s'écrit sous la forme $w = uxzyv$, où

- les chaînes x et y ne sont pas toutes deux vides
- la longueur de xzy est majorée par K
- $ux^nzy^nv \in L$ pour tout $n \geq 0$

Preuve. Si L est infini, prendre $K = m^{|V|+1}$, où m est la longueur maximale des règles de production et $|V|$ le nombre de variables d'une grammaire engendrant L . On a $m > 1$ si L est infini, donc $m^{|V|+1} > m^{|V|}$. Le raisonnement précédent permet de finir la preuve.

Si L est fini (cas où le théorème est sans intérêt) : prendre $K = \max\{ |w| : w \in L\} + 1$.

Note. Pour assurer xy non vide, une autre stratégie serait de commencer par transformer la grammaire, afin de travailler avec une grammaire équivalente sans règles vides (autres que pour l'axiome) et où l'axiome est non récursif. Avec une grammaire sous forme normale de Chomsky par exemple (cf. [1]), $m = 2$. Avec une grammaire sous forme normale de Greibach, les règles terminales sont de longueur 1, on obtient $|xzy| \leq m^{|V|}$.

Pour finir, voici un petit exemple –pathologique– qui montre que dans un arbre de dérivation, il n'y a pas nécessairement deux nœuds exploitables (xy non vide) parmi $|V|+1$ nœuds consécutifs.

Une grammaire G étant donnée, il suffit de la modifier en lui ajoutant les règles $A \to A$ (pour tout $A \in V$). Des tronçons vides de n'importe quelle hauteur peuvent être insérés entre toutes les arêtes consécutives, dans un arbre de dérivation.

Avec $G : S \to Sa \mid S \mid \varepsilon$ la chaîne a^n peut s'obtenir ainsi (pour tout k) :

$$\underbrace{S \Rightarrow \ldots \Rightarrow S \Longrightarrow}_{k \text{ fois } S\to S} Sa \underbrace{\Rightarrow \ldots \Rightarrow Sa \Longrightarrow}_{\text{puis } S\to Sa} Sa^2 \underbrace{\ldots \Rightarrow Sa^2 \ldots \Longrightarrow}_{} Sa^3 \underbrace{\Rightarrow \ldots}_{} \ldots \underbrace{\ldots \Rightarrow}_{} Sa^n \Rightarrow a^n$$

Voyons maintenant un exemple d'utilisation du critère « lemme de l'Etoile », pour prouver que certains langages ne sont <u>pas</u> engendrés par une grammaire algébrique.

5.2 Exemples d'application du Lemme de l'Etoile.

L'exemple le plus simple d'un langage algébrique non régulier est $\{a^n b^n : n \geq 0\}$. En observant, dans le lemme de l'Etoile pour les langages algébriques, ces deux segments x et y dupliqués de part et d'autre d'une partie médiane, on peut imaginer que pour obtenir un exemple de langage non algébrique, il suffirait de chaînes tripartites dans lesquelles les trois parties dépendent les unes des autres.

Voici un exemple classique de ce type, sur l'alphabet $\{a, b, c\}$.

Proposition. Le langage $L = \{ a^n b^n c^n : n \geq 0 \}$ n'est pas algébrique.

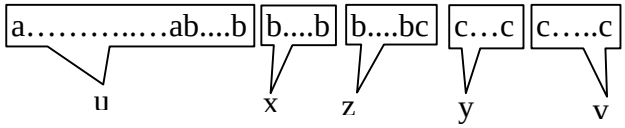
En corollaire, il existe des langages qui ne sont pas reconnaissables par un automate à pile.

Preuve. On prouve que le lemme de l'Etoile n'est pas vérifié. Quel que soit l'entier K, nous exhibons une chaîne $w \in L$, de longueur supérieure à K, qui a la propriété que pour n'importe quel découpage $w = uxzyv$ (où xy est non vide), la chaîne $uxxzyyv$ ne soit plus dans le langage.

Soit $w = a^K b^K c^K$. On écrit $w = uxzyv$ avec xy non vide.

Tout d'abord, remarquons qu'en dupliquant dans w une sous-chaîne contenant au moins deux symboles différents, la chaîne obtenue n'est certainement pas dans L puisqu'elle n'est même pas de la forme $a^i b^j c^k$.

Il reste à voir le cas où chacune des chaînes x et y est écrite avec un seul symbole répété (éventuellement zéro fois). Mais dans ce cas, l'un (au moins) des trois symboles ne figure ni dans x, ni dans y. En dupliquant x et y, ce symbole figurera K fois, donc strictement moins de fois que l'un des autres, dans la chaîne $uxxzyyv$. On est sorti de L.



(ici, le symbole a *restera en* K *exemplaires)*

Mais n'allons pas imaginer qu'il est nécessaire d'avoir trois symboles distincts dans l'alphabet ! Les exemples suivants sont à deux symboles :

Exemples 2. Le langage $L_1 = \{ a^n b^n a^n : n \geq 0 \}$ n'est pas algébrique.
$L_2 = \{ a^m b^n a^m : m \geq n \geq 0 \}$ n'est pas algébrique.
$L_3 = \{ u.u \ : \ u \in \{a, b\}^* \}$ n'est pas algébrique.

Preuves. Quel que soit l'entier K, nous avons une chaîne w, de longueur supérieure à K, qui a la propriété que la chaîne $uxxzyyv$ ne soit plus dans le langage considéré, pour n'importe quel découpage $w = uxzyv$ vérifiant $|xzy| \leq K$ et $xy \neq \varepsilon$.

La chaîne $w = a^K b^K a^K$ convient, aussi bien pour L_1 que pour L_2 d'ailleurs. Ecrivons $a^K b^K a^K = uxzyv$, avec $|xzy| \leq K$ et xy non vide. Il faut examiner toutes les dispositions possibles pour la chaîne xzy. La facteur x (resp. y) ne peut contenir deux symboles différents. En dupliquant x et y, on obtient la chaîne $uxxzyyv$ qui n'est dans L_1 dans aucun cas de figure. Dans le cas de L_2, la condition $|xzy| \leq K$ doit être exploitée (pour interdire $x = a^i$ dans le préfixe avec $y = a^i$ dans le suffixe) ; alors, $uxxzyyv \notin L_2$.

Pour L_3, le raisonnement est analogue à partir de $w = a^K b^K a^K b^K$.

Noter la différence de nature entre le langage L_3 des chaînes doublées et le langage des palindromes de longueur paire $\{ u.u^t : u \in \{a, b\}^* \}$ qui, lui, est algébrique (il est engendré par une grammaire algébrique évidente $S \rightarrow aSa \mid bSb \mid \varepsilon$).

Bien, mais même avec un seul symbole, on peut trouver des langages non algébriques, comme nous allons le voir.

Exemple 3. Le langage $L = \{ a^{2^n} : n \geq 0 \}$ n'est pas algébrique.

Preuve. Quel que soit l'entier K, considérons la chaîne a^{2^K}, dont la longueur 2^K est évidemment supérieure à K. On écrit $a^{2^K} = uxzyv$ avec xy non vide et $|xzy| \leq K$. La chaîne $uxxzyyv$ n'est pas dans L, puisque

$$2^K < |uxxzyyv| = 2^K + |xy| \leq 2^K + K < 2^{K+1} .$$

Chap. 6 Automates à pile déterministes

6.1 Définition et exemple.

Définition. Un automate à pile (à transitions généralisées) est dit *déterministe* si, quelle que soit la configuration c , il y a au plus une configuration qui dérive (en une étape) de c.

Un langage est *déterministe* s'il peut être reconnu par un automate à pile déterministe.

Remarque. A proprement parler, la définition précédente ne signifie pas qu'il y a au plus une transition qui puisse s'appliquer à la configuration donnée. Donnons un contre-exemple (un peu pathologique). En appliquant à une configuration c donnée les deux transitions formellement distinctes ([i, ε] , a , [j, M]) et ([i,P] , a , [j, MP]), on obtient au plus une configuration. Mais en présence de deux telles transitions, on peut évidemment supprimer ([i,P] , a , [j, MP]) , la première pouvant s'appliquer sans imposer de condition sur le contenu de la pile (cf. Chap. 1, §1.4. , Remarque 2).

Dorénavant, nous supposons que l'automate à pile est nettoyé de ces transitions parasites. Le critère suivant permet alors de détecter le déterminisme directement sur les transitions.

Un automate à pile (à transitions généralisées) $A = (\Sigma , E, s_o , F, \Gamma , \delta)$ est déterministe si, quels que soient l'état $i \in E$, le symbole de pile $P \in \Gamma$ et le symbole $a \in \Sigma$, il y a au plus une transition de la forme ([i,P'] , a' , [j, M]) avec $P' \in \{P , \varepsilon \}$, $a' \in \{a , \varepsilon \}$, $j \in E$, et $M \in \Gamma^*$.

S'il y a une transition ([i, ε] , ε, [j, M]), elle est l'unique transition issue de l'état i . Sinon, s'il y a une transition ([i, P] , ε, [j, M]), c'est l'unique transition issue de i qui dépile $P \in \Gamma$; s'il y a une transition ([i, ε] , a , [j, M]), c'est l'unique transition consommant le symbole $a \in \Sigma$; et s'il y a une transition qui dépile P et consomme a , elle est la seule dans son genre .

Dans un automate à pile déterministe, il y a donc unicité du cheminement : à partir d'une configuration initiale donnée avec une chaîne w à tester, la suite des transitions pouvant être successivement empruntées est déterminée de façon unique. La progression peut évidemment être bloquée avant la fin de la lecture de w . Mais à cause du déterminisme justement, il est possible de « compléter » l'automate à pile déterministe : en ajoutant un nouvel état « puits perdu » et les transitions nécessaires pour lever les blocages, on obtient un automate à pile déterministe complet, dans lequel la lecture de la chaîne w peut être poursuivie jusqu'au bout.

Comme tous les automates peuvent être effectivement transformés en automates déterministes complets (cf. [1]), il est naturel de se demander s'il est possible de trouver un procédé analogue pour les automates à pile. La réponse est non : il n'existe pas d'algorithme de « déterminisation ». Pire encore : il y a des langages algébriques qui ne sont pas déterministes.

Exemple. On peut prouver que le langage des palindromes sur $\Sigma = \{a, b\}$ n'est pas déterministe : il n'existe aucun automate à pile déterministe qui le reconnaisse.

L'idée : il est impossible de vérifier la symétrie de la chaîne testée en lisant ses symboles un à un de gauche à droite. Le centre de symétrie de la chaîne ne peut pas être trouvé

autrement qu'en essayant systématiquement tous les cas possibles. Nous reviendrons plus en détail sur cet exemple dans 6.3.

6.2 Discussion des modes de reconnaissance.

On a vu que pour les langages reconnaissables par un automate à pile, le choix du mode de reconnaissance est indifférent puisque ceux-ci sont équivalents via des transformations des automates à pile (cf. Chap. 2).

Pour les langages déterministes, il en ira autrement. Mais commençons par voir ce qui peut être sauvé, lorsque ces transformations sont appliquées dans le cadre déterministe.

i) La transformation 2.2 *1.* (à partir d'un automate à pile qui reconnaît par état acceptant), appliquée à un automate à pile $\mathcal{A}$ déterministe, donne un résultat qui n'est pas déterministe en général. Les transitions ajoutées pour vider la pile rendent le nouvel automate à pile non déterministe s'il y a dans $\mathcal{A}$ une transition issue d'un état acceptant.

A priori, l'exigence d'une pile vide dans les configurations acceptantes est donc une contrainte supplémentaire, dans le cas déterministe.

ii) Voyons les transformations 2.2 *2.* et 2.2 *3.* à partir d'un $\mathcal{A}$ déterministe qui reconnaît par état acceptant et pile vide. S'il y a, dans $\mathcal{A}$, une transition de la forme ([i, ε], a, [j, M]) issue d'un état acceptant i, on obtient un $\mathcal{A}'$ non déterministe (que a soit vide ou non). Bien entendu, dans $\mathcal{A}'$, cette transition ([i, ε], a, [j, M]) équivaut à la famille de transitions ([i, P] , a, [j, MP]) , $P \in \Gamma \cup \{\Delta\}$.

En revanche, si toutes les transitions issues des états acceptants de $\mathcal{A}$ commencent par dépiler un symbole de pile de $\mathcal{A}$, alors le résultat est déterministe. Comme les transformations locales de 1.3 et 1.4 conservent le déterminisme, on peut même arriver à un automate à pile déterministe dont la forme est celle décrite dans la Proposition de 2.3 .

Définition. Soit L un langage algébrique. Le langage minimal *Min*(L) est constitué de toutes les chaînes $w \in L$ pour lesquelles aucun préfixe propre n'appartient à L :

$$\text{Min}(L) = \{ w \in L : \text{si } w = uv \text{ avec } u \text{ et } v \text{ non vides, alors } u \notin L \}$$

Attention, le langage Min(L) n'a aucune raison d'être algébrique, a priori.

Proposition 1. Soit L un langage que reconnaît, par pile vide, un automate à pile déterministe dont toutes les transitions sont de la forme ([i, P] , a , [j, M]) avec le symbole P non vide (l'étiquette a ou la chaîne M peuvent être vides ou non).
Alors on a L = Min(L) .

Preuve. Dans un tel automate à pile :
i) à cause du déterminisme, si $w = uv \in L$ avec $u \in \text{Min}(L)$, le cheminement menant à l'acceptation de w doit être un prolongement de celui qui est associé à u
ii) comme la pile est vide au moment où u est acceptée, aucune transition ne peut plus s'appliquer (à cause de la forme des transitions).
Si la reconnaissance se fait par pile vide et état acceptant, l'argument est le même.

Définition. Un langage L est dit *préfixe* si L = Min(L) .

Remarque. La grammaire construite (cf. 3.4) à partir d'un automate à pile satisfaisant aux hypothèses de la Proposition précédente est non ambiguë.

Proposition 2 . Soit L un langage que reconnaît, par états acceptants, un automate à pile déterministe.
Alors Min(L) est algébrique et déterministe reconnu par états acceptants.

Preuve. Soit $\mathcal{A}$ déterministe, reconnaissant L par états acceptants. A cause du déterminisme, si $w = uv \in L$ avec $u \in Min(L)$, le cheminement menant à l'acceptation de w est un prolongement de celui associé à u . En supprimant toutes les transitions issues des états acceptants de $\mathcal{A}$, on obtient un automate à pile avec lequel seules les chaînes de Min(L) sont acceptées.

Mentionnons au moins les deux propriétés de fermeture suivantes pour les langages déterministes reconnus par états acceptants.

Proposition 3. i) L'intersection d'un langage déterministe et d'un langage *régulier* est un langage déterministe.

ii) Le complémentaire d'un langage déterministe est déterministe.

Preuve.
i) Tout langage régulier est reconnu par un automate déterministe (et complet). La construction faite en 4.2 Proposition i), appliquée à deux dispositifs déterministes, donne un automate à pile déterministe.
ii) Un automate à pile déterministe pouvant être complété, il reste ensuite à inverser les états acceptants et non-acceptants pour reconnaître le langage complémentaire.

A cette occasion, on rappelle que le complémentaire d'un langage algébrique n'est pas, en général, algébrique (cf. 4.2)

6.3 Des exemples parmi les palindromes.

Soit L le langage de tous les palindromes, sur {a, b}. C'est un langage algébrique (on a déjà vu une grammaire adéquate). Mais le langage Min(L) est dramatiquement réduit, parce que les préfixes (à un seul symbole) a et b sont déjà des palindromes :

$Min(L) = \{a , b , \varepsilon\}$.

On en déduit que L n'est pas déterministe « par pile vide », d'après la Proposition 1 .
Mais L est-il déterministe « par états acceptants » ? Supposons que c'est le cas.

L'intersection de L avec le langage régulier $a^*ba^*(b \cup\{\varepsilon\})a^*ba^*$ est un langage L' algébrique (cf. 4.2), et il est aussi déterministe d'après l'hypothèse (intersection d'un langage déterministe et d'un langage régulier). D'après la Proposition 2 précédente, Min(L') doit donc être un langage algébrique (et, accessoirement, déterministe).

Les éléments de L' sont les palindromes qui contiennent deux ou trois b . Ceux qui contiennent (exactement) deux b sont tous minimaux (leurs préfixes propres sont asymétriques, ou ne sont pas dans L'). Ceux qui contiennent trois b sont de la forme $a^m\, ba^n\, ba^nba^m$; ils sont minimaux lorsque $m > n \geq 0$. Par exemple, abaabaaba n'est pas minimal (préfixe propre : abaaba) alors que aababababaa est minimal.

Résumons : $Min(L') = \{ a^m\, ba^n\, ba^m : m \geq 0 ,\ n \geq 0 \} \cup \{ a^m\, ba^n\, ba^nba^m :\ m > n \geq 0 \}$.

Or, ce langage n'est pas algébrique, d'après le Lemme de l'Etoile : quel que soit K , en écrivant $a^{K+1}\, ba^K\, ba^Kba^{K+1} = uxzyv$ avec $|xzy| \leq K$ et $xy \neq \varepsilon$, la chaîne uxxzyyv est asymétrique, ou avec trop de a en partie centrale : $a^{K+1}\, ba^{K+|x|}\, ba^{K+|y|}\, ba^{K+1} \notin Min(L')$.

Nous avons abouti à une contradiction.

La proposition suivante est ainsi prouvée.

Incidemment, nous avons aussi obtenu un exemple de langage algébrique dont le langage minimal n'est pas algébrique.

Proposition. Le langage algébrique des palindromes sur $\{a, b\}$ n'est pas déterministe.

Remarque. Dans la littérature sur le sujet, on trouve généralement une preuve de cette proposition qui utilise un autre sous-langage intéressant : celui des palindromes qui sont de la forme $(ab)^*(ba)^*(ab)^*(ba)^*$. Les éléments minimaux en sont ceux qui ont la forme $(ab)^i (ba)^j (ab)^j (ba)^i$ avec $i > j$. On peut prouver que les langages algébriques sont fermés par « automorphisme inverse » (une notion que nous n'avons pas abordée dans ce cours). Dans le cas considéré, l'idée est qu'on a remplacé a par ab et b par ba , et qu'en revenant en arrière, on obtient le langage $\{ a^i b^j a^j b^i : i > j \}$. Si L était déterministe, ce dernier langage devrait être algébrique. Or ce n'est pas le cas (lemme de l'Etoile).

REFERENCES

Cours précédent, sur le site Internet *LesMathématiques.net*

[1] M.P. Muller, *Langages - Grammaires – Automates* (pdf , à télécharger)

Références externes

J. Berstel, *Transductions and context-free languages*. Teubner, 1979
J. Hopcroft ; J. Ullman, *Introduction to Automata Theory, Languages and Computation*. Addison Wesley, 1979
M. Sipser, *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.
T. Sudkamp, *Languages and Machines.* Addison Wesley.
P. Wolper. *Introduction à la calculabilité*. Dunod, Paris, 2001.

Rappel – Références des publications originelles des résultats fondamentaux mentionnés.

Introduction par S. Kleene, en 1956, de la notion de langage régulier :

Kleene, S. , *Representation of Events in Nerve Nets and Finite Automata* in Automata Studies (1956) eds. C. Shannon and J. McCarthy.

Les travaux fondamentaux de N. Chomsky sur la classification des langages et grammaires :

1. Chomsky, N., *Three models for the description of language*, IRE Transactions on Information Theory, 2 (1956), pages 113-124

2. Chomsky, N., *On certain formal properties of grammars*, Information and Control, 1 (1959), pages 91-112

L'article de Sheila Greibach introduisant, en 1965, la forme normale de Greibach, a été l'une de ses premières publications :

Greibach, S., *A New Normal-Form Theorem for Context-Free Phrase Structure Grammars,* Journal of the ACM (JACM), Volume 12 Issue 1 (1965)