Chapitre 2

Langages algébriques

Les langages algébriques, aussi appelés langages *hors contexte* (*context-free* en anglais) constituent le deuxième niveau de la hiérarchie de Chomsky. Il furent initialement introduits pour modéliser les langues naturelles. Cette approche a été relativement abandonnée même si des grammaires locales sont encore utilisées pour décrire des constructions grammaticales. Par contre, ce modèle s'est avéré particulièrement adapté pour décrire la syntaxe des langages de programmation. D'un point de vue plus pratique, des outils tels `yacc` ou `bison` permettent de construire automatiquement un analyseur syntaxique à partir d'une grammaire qui satisfait quelques hypothèses supplémentaires.

Une première partie de ce chapitre traite des grammaires qui permettent d'engendrer ces langages algébriques. Une seconde partie est consacrée aux automates à pile qui acceptent les langages algébriques. Ces automates étendent les automates finis en utilisant une mémoire externe sous la forme d'une pile. Les liens avec les systèmes d'équations sont également abordés et utilisés pour prouver le théorème de Parikh. Pour tout ce chapitre, on se reportera à [Aut87] ou à [ABB97].

Les problèmes de décidabilité et de complexité concernant les langages algébriques sont abordés dans les deux chapitres suivants. Il sera en particulier montré que l'appartenance d'un mot à un langage algébrique peut être décidée en temps polynomial alors que l'égalité de deux langages algébriques est indécidable.

L'utilisation des grammaires pour construire des analyseurs syntaxiques n'est pas abordée dans ce chapitre. Pour ces aspects plus pratiques, il est préférable de consulter des ouvrages spécialisés comme [ASU86].

2.1 Grammaires algébriques

2.1.1 Définitions et exemples

Une grammaire algébrique est en quelque sorte un système de réécriture qui engendre un ensemble de mots à partir d'un axiome. Afin de bien distinguer les lettres des mots engendrés des lettres intermédiaires, l'alphabet est partagé en lettres terminales et en lettres non terminales appelées variables. Seules les variables peuvent subir une réécriture.

Définition 2.1 (Grammaire algébrique)**.** Une *grammaire algébrique* G est un triplet (A, V, P) où A et V sont des alphabets finis et disjoints et où P est une partie finie de $V \times (A \cup V)^*$. Les symboles de A sont appelés *terminaux* et ceux de V sont appelés

non terminaux ou *variables*. Les éléments de P sont appelés *règles*. Chaque règle est une paire (X, u) où X est une variable et u est un mot sur l'alphabet $A + V$.

Les grammaires définies ici sont appelées algébriques pour les distinguer de grammaires plus générales qui sont abordées au chapitre suivant (cf. définitions 3.54 et 3.55 p. 158). Le terme *algébrique* sera justifié en montrant que les langages engendrés par les grammaires sont solutions de systèmes d'équations polynomiales (cf. proposition 2.24 p. 84). Comme toutes les grammaires considérées dans ce chapitre sont algébriques, elles seront simplement appelées grammaires dans le reste de ce chapitre.

On note $X \to u$ lorsque $(X, u) \in P$ est une règle de la grammaire. Par extension, si $(X, u_1), \ldots, (X, u_n)$ sont des règles, on écrit $X \to u_1 + \cdots + u_n$. Les éléments X et u d'une règle $X \to u$ sont respectivement appelés *membre gauche* et *membre droit* de la règle.

Exemple 2.2. Soit la grammaire $G = (A, V, P)$ définie par

$$A = \{a, b\}, \quad V = \{S\}, \quad P = \{S \to aSb + \varepsilon\}$$

Cette grammaire a deux lettres terminales a et b, une seule variable S et deux règles (S, aSb) et (S, ε). On verra que le langage engendré par cette grammaire est $\{a^n b^n \mid n \geq 0\}$.

Lorsqu'une grammaire est donnée de manière explicite, on se contente souvent de donner les règles. Les variables de la grammaire sont alors implicitement les symboles qui apparaissent comme membre gauche des règles et toutes les autres lettres sont implicitement des lettres terminales. Pour la grammaire donnée ci-dessus, on aurait pu simplement écrire $P = \{S \to aSb + \varepsilon\}$.

Définition 2.3 (Dérivation)**.** Soit $G = (A, V, P)$ une grammaire et soient u et v deux mots sur $A + V$. On dit que u se *dérive* en (ou *produit*) v et on note $u \to v$ lorsque il existe $\alpha, \beta \in (A + V)^*$ et $X \in V$ tels que :

$$u = \alpha X \beta, v = \alpha w \beta \text{ et } (X \to w) \in P$$

La dérivation est dite *gauche* (respectivement *droite*) si α (respectivement β) appartient à A^*. Ceci signifie que c'est la variable la plus à gauche (respectivement à droite) qui est réécrite.

Pour $k = 0$, on note $u \xrightarrow{k} v$ si $u = v$ et pour $k \geq 1$, on note $u \xrightarrow{k} v$ s'il existe des mots $u_1, u_2, \ldots, u_{k-1}$ tels que $u \to u_1 \to \cdots \to u_{k-1} \to v$. Plus généralement, on notera $u \xrightarrow{*} v$ s'il existe un entier $k \geq 0$ tel que $u \xrightarrow{k} v$. La relation $\xrightarrow{*}$ est donc la clôture réflexive et transitive de la relation $\to$.

Exemple 2.4. Soit G la grammaire donnée à l'exemple 2.2. La variable S peut se dériver en le mot aSb qui se dérive à nouveau en $aaSbb$, qui se dérive encore en $aaaSbbb$ qui se dérive finalement en a^3b^3 : $S \xrightarrow{*} a^3b^3$. Il est facile de voir que les mots $w \in A^*$ tels que $S \xrightarrow{*} w$ sont justement les mots de la forme $a^n b^n$ pour $n \geq 0$.

Les langages algébriques sont aussi appelés langages hors contexte parce que lors d'une dérivation, le remplacement de la variable X par w dans le mot $u = \alpha X \beta$ est indépendant des contextes α et β. Il existe aussi des grammaires contextuelles (cf. section 3.7.2) dont les membres gauches de règle ne sont plus simplement des variables mais des mots qui permettent de spécifier le contexte de remplacement d'une variable.

Définition 2.5 (Langage engendré)**.** Soit $G = (A, V, P)$ une grammaire et soit u un mot sur $(A+V)^*$. On note respectivement $\widehat{L_G}(u)$ et $L_G(u)$ les langages $\{v \in (A+V)^* \mid u \xrightarrow{*} v\}$ et $\widehat{L_G}(u) \cap A^*$

Dans une grammaire G, une variable S_0 appelée *axiome* est parfois distinguée. Le *langage engendré* par la grammaire est alors implicitement le langage $L_G(S_0)$.

Définition 2.6 (Langage algébrique)**.** Un langage est dit *algébrique* s'il peut être engendré par une grammaire, c'est-à-dire s'il est égal à $L_G(S)$ pour une variable S d'une grammaire G.

Exemple 2.7. Soit G la grammaire donnée à l'exemple 2.2. Le langage $L_G(S)$ est égal à $\{a^n b^n \mid n \geq 0\}$ qui est donc un langage algébrique.

Grâce au lemme fondamental 2.11 et à la clôture des langages algébriques par produit (cf. proposition 2.50), on constate que $L_G(u)$ est encore algébrique pour tout mot $u \in (A + V)^*$. La notation L_G peut être étendue aux langages en posant $L_G(K) = \bigcup_{u \in K} L_G(u)$ pour $K \subset (A + V)^*$. La clôture des langages algébriques par substitution algébrique (cf. proposition 2.52) montre que $L_G(K)$ reste algébrique quand K est algébrique.

Exemple 2.8. Quelques exemples de grammaires et de langages algébriques.

1. Le langage a^*b est engendré par la grammaire $S \to aS + b$. Il sera montré que tous les langages rationnels sont algébriques.
2. Grammaire
$$\begin{cases} A = \{a, b, c\} \\ S \to AS + \varepsilon \\ A \to AB + a \\ B \to BC + b \\ C \to CA + c \end{cases}$$
3. Langage de Dyck sur n paires de parenthèses.
$$\begin{cases} A_n = \{a_1, \ldots, a_n, \bar{a}_1, \ldots, \bar{a}_n\} \\ S \to ST + \varepsilon \\ T \to a_1 S \bar{a}_1 + \cdots + a_n S \bar{a}_n \end{cases}$$
Les langages $D_n = L_G(T)$ et $D_n^* = L_G(S)$ sont respectivement appelés *langage de Dyck primitif* et *langage de Dyck*. Si chaque lettre a_i est vue comme une parenthèse ouvrante et la lettre $\bar{a}_i$ comme la parenthèse fermante correspondante, les mots du langages de Dyck sont les mots bien parenthésés. De façon équivalente, ce sont les mots qui se réduisent au mot vide en appliquant les réductions de la forme $a_i \bar{a}_i \to \varepsilon$.
4. *Langage de Luckasiewicz*
$$S \to aSS + \bar{a}$$
Le langage Ł $= L_G(S)$ est l'ensemble des mots w tels que $|w|_{\bar{a}} = |w|_a + 1$ et $|u|_{\bar{a}} \leq |u|_a$ pour tout préfixe (propre) u de w. On a l'égalité Ł $= D_1^* \bar{a}$.

5. *Langage de Goldstine*

$$L = \{a^{n_0}ba^{n_1}b\cdots a^{n_k}b \mid k \geq 0 \text{ et } \exists\, j \geq 0, n_j \neq j\}$$

Le langage L est égal à $L_G(S)$ où la grammaire G est constituée des règles suivantes.

$$\begin{aligned} T_0 &\to aT_0 + \varepsilon \\ T_1 &\to T_0 b \\ T_2 &\to T_1T_2 + \varepsilon \\ T_3 &\to T_1T_3a + T_1T_2 + aT_0 \\ S &\to T_3bT_2 \end{aligned}$$

On vérifie facilement que $L_G(T_0) = a^*$, $L_G(T_1) = a^*b$, $L_G(T_2) = (a^*b)^*$, $L_G(T_3) = \{a^{n_0}ba^{n_1}b\cdots a^{n_k} \mid k \geq 0 \text{ et } n_k \neq k\}$ et $L_G(S) = L$.

Exercice 2.9. Montrer que l'ensemble des palindromes écrits sur un alphabet A est un langage algébrique. Montrer que le complémentaire de ce langage est aussi algébrique.

Solution. Pour simplifier, on suppose que l'alphabet est $A = \{a, b\}$. L'ensemble des palindromes sur cet alphabet est engendré par la grammaire $S \to aSa + bSb + a + b + \varepsilon$. Le complémentaire est engendré par la grammaire $S \to aSa + bSb + aTb + bTa,\ \ T \to aTa + aTb + bTa + bTb + a + b + \varepsilon$.

Exercice 2.10. Soit A un alphabet ne contenant pas le symbole #. Montrer que les deux langages

$$\begin{aligned} L_1 &= \{ww' \mid w, w' \in A^*, w \neq w' \text{ et } |w| = |w'|\}, \\ L_2 &= \{w\#w' \mid w, w' \in A^* \text{ et } w \neq w'\} \end{aligned}$$

sont algébriques.

Les résultats de cet exercice sont à comparer avec ceux de l'exercice 2.48.

Solution. On suppose pour simplifier que A est l'alphabet $\{a, b\}$. La preuve se généralise sans problème à un alphabet plus large. Le langage L_1 est égal à $L_aL_b \cup L_bL_a$ où les langages L_a et L_b sont respectivement égaux à $\{uav \mid |u| = |v|\}$ et $\{ubv \mid |u| = |v|\}$. On tire facilement une grammaire de cette expression.

Le langage L_2 est égal à l'union non disjointe des langages $L_<$, $L_>$, L_{ab} et L_{ba} définis par les formules suivantes.

$$\begin{aligned} L_< &= \{w\#w' \mid |w| < |w'|\} && \text{et symétriquement pour } L_> \\ L_{ab} &= \{uav\#u'bv' \mid |u| = |u'|\} && \text{et symétriquement pour } L_{ba} \end{aligned}$$

Le langage $L_<$ est égal à $L_G(S)$ où la grammaire G est donnée par

$$\begin{cases} S \to aSa + aSb + bSa + bSb + Ta + Tb \\ T \to Ta + Tb + \#. \end{cases}$$

Le langage L_{ab} est égal à $L_{G'}(S_0)$ où la grammaire G' est donnée par

$$\begin{cases} S_0 \to SbT \\ S \to aSa + aSb + bSa + bSb + aT\# \\ T \to Ta + Tb + \varepsilon. \end{cases}$$

Le lemme suivant énonce un résultat très simple mais il permet de formaliser beaucoup de preuves en raisonnant sur la longueur des dérivations. Il établit que les dérivations partant d'un mot $u = u_1u_2$ se décomposent en des dérivations partant de u_1 et des dérivations partant de u_2 et que ces deux types de dérivations sont indépendantes.

Lemme 2.11 (Fondamental)**.** *Soit $G = (A, V, P)$ une grammaire et soit u et v deux mots sur $(A + V)^*$. On suppose que u se factorise $u = u_1u_2$. Il existe une dérivation $u \xrightarrow{k} v$ de longueur k si et seulement si v se factorise $v = v_1v_2$ et s'il existe deux dérivations $u_1 \xrightarrow{k_1} v_1$ et $u_2 \xrightarrow{k_2} v_2$ où $k = k_1 + k_2$.*

Une manière un peu moins précise d'énoncer le lemme fondamental est d'écrire que pour tous mots u et v sur $(A + V)^*$, on a l'égalité

$$\widehat{L_G}(uv) = \widehat{L_G}(u)\widehat{L_G}(v).$$

Preuve. Il est aisé de combiner les dérivations $u_1 \xrightarrow{k_1} v_1$ et $u_2 \xrightarrow{k_2} v_2$ pour obtenir une dérivation $u_1u_2 \xrightarrow{k} v_1v_2$ de longueur $k_1 + k_2$. La réciproque se montre par induction sur k. □

2.1.2 Grammaires réduites

Les manipulations sur les grammaires algébriques sont souvent facilitées par des règles d'une forme particulière. Il existe plusieurs formes normales de grammaires plus ou moins élaborées. Nous commençons par quelques formes normales très faciles à obtenir. Nous insistons sur le côté calculatoire pour bien montrer que ces formes normales peuvent être effectivement calculées.

La notion de grammaire réduite est l'analogue pour les grammaires de la notion d'automate émondé. De manière intuitive, une grammaire est réduite si elle ne contient pas de variable trivialement inutile.

Définition 2.12 (Grammaire réduite)**.** Une grammaire $G = (A, V, P)$ est dite *réduite* pour $S_0 \in V$ si

1. pour tout $S \in V$, $L_G(S) \neq \varnothing$,
2. pour tout $S \in V$, il existe $u, v \in (A + V)^*$ tels que $S_0 \xrightarrow{*} uSv$.

La première condition signifie que toute variable peut produire un mot et la seconde condition signifie que toute variable peut être atteinte à partir de la variable S_0.

Proposition 2.13 (Réduction)**.** *Pour toute grammaire $G = (A, V, P)$ et toute variable $S_0 \in V$, il existe une grammaire G' réduite pour S_0' telle que $L_G(S_0) = L_{G'}(S_0')$.*

Preuve. Pour réduire, on applique successivement les deux étapes suivantes. Il est à noter que l'ordre des étapes est important.

La première étape consiste à supprimer des variables S telles que $L_G(S) = \varnothing$. On définit par récurrence une suite $(U_n)_{n\geq 0}$ d'ensembles de symboles de $A + V$ par

$$\begin{aligned} U_0 &= A, \\ U_{n+1} &= U_n \cup \{S \in V \mid S \to w \text{ et } w \in U_n^*\} \quad \text{pour } n \geq 0. \end{aligned}$$

Puisque l'ensemble $A + V$ est fini et que la suite $(U_n)_{n\geq 0}$ est croissante, celle-ci est constante à partir d'un certain rang. Soit U l'ensemble $\bigcup_{n\geq 0} U_n$. On prouve par récurrence sur la longueur d'une plus petite dérivation que $U \cap V$ est l'ensemble $\{S \mid L_G(S) \neq \varnothing\}$. On supprime alors les variables qui n'appartiennent pas à $U \cap V$ et toutes les règles où ces variables apparaissent.

La seconde étape consiste à supprimer toutes les variables inaccessibles à partir de l'axiome S_0. On définit par récurrence une suite $(W_n)_{n\geq 0}$ d'ensembles de variables par

$$\begin{aligned} W_0 &= \{S_0\}, \\ W_{n+1} &= W_n \cup \{S \in V \mid \exists\ S' \in W_n, S' \to uSv \text{ avec } u, v \in (A+V)^*\}. \end{aligned}$$

Puisque V est fini et que la suite $(W_n)_{n\geq 0}$ est croissante, celle-ci est constante à partir d'un certain rang. On montre que l'ensemble $W = \bigcup_{n\geq 0} W_n$ est égal à l'ensemble $\{S \mid S_0 \xrightarrow{*} uSv \text{ avec } u, v \in A^*\}$ en utilisant le fait que $L_G(S)$ est non vide pour toute variable S. Ainsi, W ne contient que les variables accessibles depuis S_0. La grammaire réduite est obtenue en ne conservant que les variables de W et que les règles faisant uniquement intervenir ces variables. □

2.1.3 Grammaires propres

Nous continuons de mettre les grammaires sous des formes plus faciles à manipuler en éliminant certaines règles souvent gênantes.

Définition 2.14 (Grammaire propre)**.** Une grammaire $G = (A, V, P)$ est *propre* si elle ne contient aucune règle de la forme $S \to \varepsilon$ ou de la forme $S \to S'$ pour $S, S' \in V$.

Exemple 2.15. la grammaire $G = \{S \to aSb + \varepsilon\}$ n'est pas propre puisque $S \to \varepsilon$. Par contre, la grammaire $G' = \{S \to aSb + ab\}$ est propre et elle engendre les mêmes mots à l'exception du mot vide.

La proposition suivante établit que toute grammaire peut être rendue propre en perdant éventuellement le mot vide.

Proposition 2.16. *Pour toute grammaire $G = (A, V, P)$ et tout $S \in V$, il existe une grammaire propre $G' = (A, V', P')$ et $S' \in V'$ telle que $L_{G'}(S') = L_G(S) \setminus \{\varepsilon\}$.*

La preuve de la proposition donne une construction effective d'une grammaire propre. Elle utilise la notion de substitution qui est maintenant définie.

Définition 2.17 (Substitution)**.** Soient A et B deux alphabets. Une *substitution* de A^* dans B^* est un morphisme $\sigma : A^* \to \mathfrak{P}(B^*)$ où $\mathfrak{P}(B^*)$ est muni du produit des langages.

Preuve. En utilisant la proposition précédente, on peut supposer que la grammaire G est réduite. La construction se décompose en deux étapes décrites ci-dessous. L'ordre des étapes a de nouveau son importance.

La première étape consiste à supprimer les règles de la forme $S \to \varepsilon$. On commence par calculer l'ensemble $U = \{S \mid S \xrightarrow{*} \varepsilon\}$ des variables qui produisent le mot vide. On définit par récurrence la suite $(U_n)_{n\geq 0}$ d'ensembles de variables par

$$\begin{aligned} U_0 &= \{S \mid S \to \varepsilon\}, \\ U_{n+1} &= U_n \cup \{S \mid S \to w \text{ et } w \in U_n^*\}. \end{aligned}$$

La suite $(U_n)_{n\geq 0}$ est constante à partir d'un certain rang. On montre par récurrence que $U = \bigcup_{n\geq 0} U_n$ est l'ensemble $\{S \mid S \xrightarrow{*} \varepsilon\}$ des variables qui produisent le mot vide. On introduit la substitution σ de $(A+V)^*$ dans $(A+V)^*$ par

$$\sigma(a) = a \quad \text{pour } a \in A$$
$$\sigma(S) = \begin{cases} S + \varepsilon & \text{si } S \xrightarrow{*} \varepsilon \\ S & \text{sinon} \end{cases} \quad \text{pour } S \in V.$$

La première étape consiste alors à procéder aux deux transformations suivantes sur la grammaire G. L'idée intuitive est de remplacer par le mot vide dans les membres droits de règles les variables qui produisent le mot vide.

1. Supprimer les règles $S \to \varepsilon$.
2. Ajouter toutes les règles $S \to u$ où $S \to w$ est une règle et $u \in \sigma(w)$.

On montre par récurrence sur la longueur des dérivations que dans la nouvelle grammaire, chaque variable engendre les même mots au mot vide près.

La seconde étape consiste à supprimer les règles de la forme $S \to S'$ où S et S' sont deux variables. Puisqu'aucune variable ne produit le mot vide, on a une dérivation $S \xrightarrow{*} S'$ si et seulement si il existe des variables $S_1, \ldots, S_n$ telles que $S_1 = S$, $S_n = S'$ et $S_i \to S_{i+1}$ pour tout $1 \leq i \leq n-1$. La relation $\xrightarrow{*}$ restreinte aux variables est donc la clôture transitive et réflexive de la relation $\to$ restreinte aux variables.

La seconde étape consiste alors à procéder aux deux transformations suivantes sur la grammaire G. L'idée intuitive est de remplacer par une seule dérivation une suite de dérivations $S_i \to S_{i+1}$ suivie d'une dérivation $S_n \to w$ où $w \notin V$.

1. Supprimer les règles $S \to S'$.
2. Ajouter les règles $S \to w$ où $S \xrightarrow{*} S'$, et $S' \to w$ est une règle avec $w \notin V$.

On vérifie sans difficulté que la grammaire obtenue est propre et qu'elle engendre le même langage.

Une autre méthode consiste à d'abord quotienter l'ensemble V par la relation d'équivalence $\equiv$ définie par

$$S \equiv S' \iff^{\text{def}} (S \xrightarrow{*} S' \text{ et } S' \xrightarrow{*} S).$$

La grammaire obtenue engendre le même langage car $S \equiv S'$ implique que $L_G(S) = L_G(S')$. La relation $\xrightarrow{*}$ devient alors un ordre sur le quotient. On supprime alors les règles $S \to S'$ où S est maximal pour cet ordre en ajoutant les règles $S' \to w$ pour chaque règle $S \to w$. Le processus est itéré avec les nouvelles variables devenues maximales pour l'ordre $\xrightarrow{*}$. □

Exemple 2.18. La grammaire $\{S \to aS\bar{a}S + \varepsilon\}$ est transformée en la grammaire $\{S \to aS\bar{a}S + aS\bar{a} + a\bar{a}S + a\bar{a}\}$ par la première étape de la preuve.

Exercice 2.19. Montrer qu'il existe un algorithme pour déterminer pour une grammaire G et pour une variable S de G si le langage $L_G(S)$ est infini ou non.

Solution. On suppose la grammaire $G = (A, V, P)$ propre. On construit alors le graphe orienté H dont l'ensemble des sommets est V et l'ensemble des arêtes est l'ensemble des paires (S, T) telles qu'il existe une règle $S \to w$ avec $w = uTv$ et $u, v \in (A+V)^*$ dans G. Il n'est pas difficile de voir que $L_G(S)$ est infini si et seulement si il existe un cycle dans H qui soit accessible à partir de S.

2.1.4 Forme normale quadratique

Nous terminons par la forme normale quadratique appelée aussi forme normale de Chomsky. Celle-ci permet en outre de savoir si un mot est engendré par une grammaire.

Définition 2.20 (Forme normale quadratique)**.** Une grammaire est en *forme normale quadratique* si toutes ses règles sont d'une des formes suivantes :
- $S \to S_1 S_2$ où $S_1, S_2 \in V$
- $S \to a$ où $a \in A$

Il faut remarquer qu'une grammaire en forme quadratique est nécessairement propre. En particulier, elle ne peut pas engendrer le mot vide.

Proposition 2.21 (Forme normale quadratique)**.** *Pour toute grammaire $G = (A, V, P)$ et tout $S \in V$, il existe une grammaire en forme normale quadratique $G' = (A, V', P')$ et $S' \in V'$ telles que $L_{G'}(S') = L_G(S) \setminus \{\varepsilon\}$.*

Preuve. Grâce à la proposition 2.16, on peut supposer que la grammaire G est propre. Ensuite, la construction se décompose à nouveau en deux étapes.

La première étape consiste à se ramener à une grammaire où tous les membres droits de règles sont soit une lettre terminale soit un mot formé uniquement de variables, c'est-à dire d'une des formes suivantes.
- $S \to a$ où $a \in A$
- $S \to S_1 S_2 \cdots S_n$ où $S_1, \ldots, S_n \in V$

Soit $V' = \{V_a \mid a \in A\}$ un ensemble de nouvelles variables en bijection avec A. Soit G' la grammaire $(A, V \cup V', P')$ où P' est l'ensemble $\{V_a \to a \mid a \in A\} \cup \{S \to \sigma(w) \mid S \to w \in P\}$ et où σ est la substitution définie par $\sigma(S) = S$ pour $S \in V$ et $\sigma(a) = V_a$ pour $a \in A$.

Dans la seconde étape, les règles $S \to S_1 \cdots S_n$ avec $n > 2$ sont supprimées et remplacées par d'autres nouvelles règles. Dans ce but, on introduit les nouvelles variables $S'_1, \ldots, S'_{n-1}$ et on remplace $S \to S_1 \cdots S_n$ par les n règles suivantes.

$$\begin{cases} S \to S_1 S'_2 \\ S'_i \to S_i S'_{i+1} \text{ pour } 2 \leq i < n-1 \\ S'_{n-1} \to S_{n-1} S_n \end{cases}$$

□

Une conséquence de la forme normale de Chomsky est qu'il existe un algorithme pour savoir si un mot donné est engendré par une grammaire également donnée. On a déjà vu qu'il est possible de savoir si une grammaire engendre le mot vide. Dans une grammaire en forme quadratique, chaque dérivation remplace une variable par une lettre terminale ou augmente le nombre d'occurrences de variables dans le mot. Il s'ensuit qu'une dérivation d'une variable à un mot w formé de lettres terminales est de longueur au plus $2|w|$. Pour savoir si une grammaire G engendre un mot w non vide, on remplace d'abord la grammaire G par une grammaire équivalente G' en forme normale quadratique puis on examine toutes les dérivations de longueur au plus $2|w|$ de G'. Cet algorithme n'est pas très efficace car le nombre de dérivations est exponentiel. La transformation d'une grammaire quelconque en une grammaire en forme normale quadratique peut aussi

beaucoup augmenter la taille de la grammaire. On verra qu'il existe l'algorithme de Cocke, Kasami et Younger pour déterminer en temps polynomial si un mot est engendré par une grammaire.

2.2 Systèmes d'équations

On va voir que les langages algébriques sont les solutions des systèmes polynomiaux, ce qui justifie la terminologie. On utilisera cette approche pour montrer le théorème de Parikh qui établit que tout langage algébrique est commutativement équivalent à un langage rationnel.

2.2.1 Substitutions

À une grammaire, on peut associer un système d'équations en langages. Nous commençons par donner un exemple. Soit la grammaire $G = (A, V, P)$ avec $A = \{a, b\}$, $V = \{X_1, X_2\}$ et P contenant les deux règles suivantes :

$$\begin{cases} X_1 \to X_1X_2 + \varepsilon \\ X_2 \to aX_2 + b. \end{cases}$$

On associe à cette grammaire le système

$$\begin{cases} L_1 = L_1L_2 + \varepsilon \\ L_2 = aL_2 + b \end{cases}$$

en les langages L_1 et L_2 sur l'alphabet A.

On introduit quelques notations pratiques qui aident à formaliser ceci. Soit $G = (A, V, P)$ une grammaire dont l'ensemble des variables est $V = \{X_1, \ldots, X_n\}$ et soit $L = (L_1, \ldots, L_n)$ un n-uplet de langages sur A. On définit successivement des opérations de substitution par

$$\begin{aligned} \varepsilon(L) &= \{\varepsilon\} \\ a(L) &= \{a\} \text{ pour } a \in A \\ X_i(L) &= L_i \\ xy(L) &= x(L)y(L) \text{ pour } x, y \in (A + V)^* \\ K(L) &= \bigcup_{w \in K} w(L) \text{ pour } K \subseteq (A + V)^* \end{aligned}$$

On notera la cohérence dans le cas particulier $L = (X_1, \ldots, X_n)$ avec la notation $\alpha(X_1, \ldots, X_n)$ marquant une simple dépendance en $X_1, \ldots, X_n$.

Le point important est de voir que l'on procède bel et bien à une substitution des X_i par les L_i correspondants.

2.2.2 Système d'équations associé à une grammaire

Même si l'exemple permet amplement de deviner la définition, nous donnons la définition précise du système d'équations associé à une grammaire.

Définition 2.22. Soit $G = (A, V, P)$ une grammaire où $V = \{X_1, \ldots, X_n\}$. Le système associé à G est le système $\mathcal{S}(G)$ formé par les n équations

$$L_i = \sum_{X_i \to w} w(L) \quad \text{pour } 1 \leq i \leq n$$

en les variables $L_1, \ldots, L_n$.

Pour une grammaire G dont l'ensemble des variables est $\{X_1, \ldots, X_n\}$, on note L_G le n-uplet $(L_G(X_1), \ldots, L_G(X_n))$ formés des langages engendrés par les différentes variables. La proposition suivante est essentiellement une reformulation du lemme fondamental 2.11 (p. 79).

Proposition 2.23. *Pour tout mot $w \in (A + V)^*$, on a*

$$L_G(w) = w(L_G).$$

Preuve. Le résultat est déjà vrai sur les symboles de base ε, a et X_i :

$$\begin{aligned} L_G(\varepsilon) &= \varepsilon = \varepsilon(L_G) \\ L_G(a) &= a = a(L_G) \\ L_G(X_i) &= X_i(L_G). \end{aligned}$$

On écrit $w = w_1 \ldots w_n$ où $w_1, \ldots, w_n$ sont les lettres de w et on applique les règles de substitution données ci-dessus.

$$\begin{aligned} L_G(w) &= L_G(w_1 \ldots w_n) \\ &= L_G(w_1) \ldots L_G(w_n) \\ &= w_1(L_G) \ldots w_n(L_G) \\ &= w_1 \ldots w_n(L_G) \\ &= w(L_G). \end{aligned}$$

□

Voyons à présent le lien entre solutions du système associé et langages engendrés par les X_i.

2.2.3 Existence d'une solution pour $\mathcal{S}(G)$

La relation essentielle entre la grammaire et son système associé est que les langages engendrés forment une solution du système. Cette solution est en outre la plus petite comme l'établit la proposition suivante.

Proposition 2.24 (Minimalité des langages engendrés)**.** *Soit $G = (A, V, P)$ une grammaire avec $V = \{X_1, \ldots, X_n\}$. Alors L_G est la solution minimale (pour l'inclusion composante par composante) du système $\mathcal{S}(G)$ associé à G.*

La preuve de la proposition est basée sur le lemme suivant.

Lemme 2.25. *Soit $G = (A, V, P)$ une grammaire et soit L une solution du système $\mathcal{S}(G)$. Si les deux mots w et w' sur $(A+V)^*$ vérifient $w \xrightarrow{*} w'$, alors $w(L) \supset w'(L)$.*

Preuve. Il suffit de montrer le résultat pour une seule dérivation. Le cas général se prouve par récurrence sur la longueur de la dérivation. Supposons que $w = uX_iv \to uzv = w'$ où $X_i \to z$ est une règle de la grammaire. Puisque L est solution de $\mathcal{S}(G)$, on a l'égalité $L_i = \sum_{X_i \to \gamma} \gamma(L)$ et donc l'inclusion $L_i \supset z(L)$. Il en découle que $w(L) \supset w'(L)$. □

On peut maintenant procéder à la preuve de la proposition.

Preuve. On vérifie d'abord que L_G est bien une solution de $\mathcal{S}(G)$:

$$\begin{aligned} L_G(X_i) &= \sum_{X_i \to \alpha} L_G(\alpha) \\ &= \sum_{X_i \to \alpha} \alpha(L_G) \end{aligned}$$

d'après la proposition 2.23.

On vérifie ensuite que c'est la solution minimale. Soit $L = (L_1, \ldots, L_n)$ une solution de $\mathcal{S}(G)$. Pour tout mot $w \in L_G(X_i)$, on a par définition une dérivation $X_i \xrightarrow{*} w$ et par le lemme précédent l'inclusion $w(L) \subset X_i(L)$. Comme $w \in A^*$, on a $w(L) = w$ et donc $w \in L_i$. Ceci prouve l'inclusion $L_G(X_i) \subset L_i$ pour tout $1 \leq i \leq n$. □

La proposition précédente justifie a posteriori la terminologie des langages algébriques. Ils sont les solutions minimales des systèmes d'équations polynomiales.

Bien que la proposition nous donne toujours l'existence d'une solution de $\mathcal{S}(G)$, l'unicité est généralement fausse. Soit par exemple la grammaire ayant les deux règles $X \to XX + \varepsilon$. Le système associé est juste constitué de l'équation $L = LL + \varepsilon$. Une solution minimale est $L_G(X) = \{\varepsilon\}$, mais tous les langages de la forme K^* pour $K \subseteq A^*$ sont également solutions. La proposition suivante donne des conditions pour avoir l'unicité.

2.2.4 Unicité des solutions propres

Le prix à payer pour avoir l'unicité d'une solution est de se restreindre aux grammaire propres et par conséquent aux solutions propres.

Définition 2.26 (Solution propre)**.** Une solution L de $\mathcal{S}(G)$ est dite *propre* si tous les L_i sont propres, c'est-à-dire ne contiennent pas le mot vide ε.

Proposition 2.27 (Unicité des solutions propres)**.** *Soit G une grammaire propre. Alors L_G est l'unique solution* propre *de $\mathcal{S}(G)$.*

Preuve. Puisque G est propre, L_G est propre, et la proposition précédente nous dit que L_G est une solution de $\mathcal{S}(G)$. Ainsi L_G est une solution propre.

Pour un entier l, on introduit la relation $\overset{l}{\sim}$ qui capture le fait que deux langages coïncident pour les mots de longueur inférieure à l. Pour deux langages K et K' et un entier l, on note $K \overset{l}{\sim} K'$ si l'égalité

$$\{w \in K \mid |w| \leq l\} = \{w \in K' \mid |w| \leq l\},$$

est vérifiée. Cette notation est étendue aux n-uplets composante par composante. Soient $L = (L_1, \ldots, L_n)$ et $L' = (L'_1, \ldots, L'_n)$ deux solutions propres de G. On va montrer par récurrence sur l que $L \overset{l}{\sim} L'$ pour tout entier $l \geq 0$ ce qui prouve que $L = L'$.

Pour $1 \leq i \leq n$, on introduit le langage fini $S_i = \{w \mid X_i \to w\}$ de sorte que le système $\mathcal{S}(G)$ s'écrit maintenant

$$L_i = S_i(L) \quad \text{pour } 1 \leq i \leq n.$$

Comme L et L' sont propres, aucune de leurs composantes ne contient le mot vide et on a $L \overset{0}{\sim} L'$.

Supposons $L \overset{l}{\sim} L'$ pour $l \geq 0$ et montrons $L \overset{l+1}{\sim} L'$, c'est-à-dire

$$L_i \overset{l+1}{\sim} L'_i \quad \text{pour } 1 \leq i \leq n$$

ou encore

$$S_i(L) \overset{l+1}{\sim} S_i(L') \quad \text{pour } 1 \leq i \leq n.$$

Soit donc $1 \leq i \leq n$ fixé, et soit $w \in S_i$ (c'est-à-dire un mot tel que $X_i \to w$), que l'on écrit $w = w_1 \ldots w_p$ où chaque w_j est soit une lettre de A, soit une variable $X_k \in V$. L'entier p est non nul car la grammaire G est propre.

Soit ensuite $u \in w(L)$ de longueur inférieure à $l+1$. Il se décompose en $u = u_1 \cdots u_p$ où $u_j = w_j$ si w_j appartient à A et $u_j \in L_k$ si $w_j = X_k$. Comme la solution est propre, chaque mot u_j est non vide. Il s'ensuit que chaque mot u_j vérifie $|u_j| \leq l$. Sinon on aurait $p = 1$ et la grammaire G contiendrait une règle $X_i \to X_k$. Ceci est exclu car la grammaire G est propre.

Par hypothèse de récurrence, on en déduit que $u_j \in L'_k$ si $w_j = X_k$ et que u appartient aussi à $w(L')$. On a montré l'inclusion $L_i \subseteq L'_i$ pour tout $1 \leq i \leq n$. Les inclusions inverses sont aussi vérifiées par symétrie. Ceci montre que $L \overset{l+1}{\sim} L'$ et termine la preuve que $L = L'$. □

2.2.5 Théorème de Parikh

On montre ici que tout langage algébrique est commutativement équivalent à un langage rationnel.

Définition 2.28. Deux mots w et w' sont dits *anagrammes* s'il existe n lettres $a_1, \ldots, a_n$ et une permutation σ de $\{1, \ldots, n\}$ telle que $w = a_1 \ldots a_n$ et $w' = a_{\sigma(1)} \ldots a_{\sigma(n)}$. L'ensemble des anagrammes d'un mot w est noté $\overline{w}$.

Pour un langage L, on appelle *image commutative* et on note $\overline{L}$ l'ensemble $\{\overline{w} \mid w \in L\}$ des classes des mots de L. Deux langages L et M sont dits *commutativement équivalents* si $\overline{L} = \overline{M}$. Pour tous langages L et M, les deux égalités suivantes sont bien sûr vérifiées.

$$\overline{L + M} = \overline{L} + \overline{M} \quad \text{et} \quad \overline{LM} = \overline{ML}.$$

Nous sommes maintenant en mesure d'énoncer le théorème de Parikh.

Théorème 2.29 (Parikh)**.** *Tout langage algébrique L est commutativement équivalent à un langage rationnel R ($\overline{L} = \overline{R}$).*

Une conséquence du théorème est que sur un alphabet unaire, c'est-à-dire avec une seule lettre, tous les langages algébriques sont rationnels. Par contre, dès que l'alphabet contient deux lettres, il existe des langages algébriques commutatifs (c'est-à-dire clos par passage à un anagramme) qui ne sont pas rationnels. Un exemple est le langage $\{w \mid |w|_a = |w|_b\}$ des mots ayant autant d'occurrences de a que d'occurrences de b.

Les grandes étapes de la preuve sont les suivantes. On associe à une grammaire un système d'équations en commutatif et on montre que les langages engendrés par une grammaire propre sont l'unique solution du système. On montre de manière indépendante que tout système en commutatif admet aussi une solution rationnelle. Une autre preuve plus élémentaire se trouve en [Koz97, p. 201].

L'exemple suivant illustre l'énoncé du théorème.

Exemple 2.30. Au langage algébrique $L = \{a^n b^n \mid n \geq 0\}$, on peut par exemple associer le langage rationnel $R = (ab)^*$ tel que $\overline{L} = \overline{R}$.

2.2.6 Systèmes d'équations en commutatifs

Nous montrons ici que les langages algébriques engendrés par une grammaire propre sont la solution unique du système commutatif associé à la grammaire.

Soit $G = (A, V, P)$ une grammaire dont l'ensemble des variables est $V = \{X_1, \ldots, X_n\}$ et soit $L = (L_1, \ldots, L_n)$ un n-uplet de langages sur A. Soient u et v deux mots sur l'alphabet $A+V$. Si $\overline{u} = \overline{v}$ alors $\overline{u(L)} = \overline{v(L)}$. Cette propriété s'étend aisément aux langages. Si les deux langages J et K sur $A + V$ vérifient $\overline{J} = \overline{K}$ alors l'égalité $\overline{J(L)} = \overline{K(L)}$ est vérifiée.

Soit $G = (A, V, P)$ une grammaire dont on suppose par commodité que l'ensemble des variables est $\{X_1, \ldots, X_n\}$. On associe à cette grammaire un système d'équations $\overline{\mathcal{S}(G)}$ obtenu en considérant le système $\mathcal{S}(G)$ pour les langages de la forme $\overline{K}$. Une solution de $\overline{\mathcal{S}(G)}$ est donc un un n-uplet $L = (L_1, \ldots, L_n)$ de langages tel que

$$\overline{L}_i = \sum_{X_i \to w} \overline{w(L)} \quad \text{pour } 1 \leq i \leq n.$$

Il s'agit d'établir la proposition suivante, ou plutôt d'établir dans le cas commutatif le résultat de la proposition 2.27.

Proposition 2.31 (Unicité des solutions propres en commutatif)**.** *Soit G une grammaire propre. Alors $\overline{L_G}$ est l'unique solution* propre *de $\overline{\mathcal{S}(G)}$.*

La proposition précédente contient un léger abus de langage. Conformément à la définition, c'est L_G et non pas $\overline{L_G}$ qui est la solution de $\overline{\mathcal{S}(G)}$. On a employé $\overline{L_G}$ pour insister sur le fait que l'unicité doit être entendue à image commutative près. Ceci signifie que si L est une solution de $\overline{\mathcal{S}(G)}$, alors $\overline{L_G} = \overline{L}$.

Preuve. La preuve de la proposition 2.27 peut être reprise *mutatis mutandis*. En effet l'argument essentiel de cette preuve est un argument de longueur qui reste valable dans le cas commutatif. □

2.2.7 Solutions rationnelles des systèmes commutatifs

Nous montrons ici que tout système d'équations en commutatif associé à une grammaire admet une solution rationnelle. Pour cela, on considère des grammaires généralisées où l'ensemble des productions de chaque variable n'est plus un ensemble fini mais un ensemble rationnel de mots.

Définition 2.32. Une *grammaire étendue* G est un triplet (A, V, P) où A et V sont des alphabets finis et disjoints et où P est une partie de $V \times (A + V)^*$ telle que pour tout $X \in V$, l'ensemble $\{w \mid (X, w) \in P\}$ est rationnel.

Les notions de dérivation et de mot engendré se généralisent aux grammaires étendues. Il est facile de voir que l'ensemble des mots engendrés par une grammaire étendue est algébrique. Il suffit pour cela de rajouter des variables (autant que d'états) qui simulent chacun des automates acceptant les langages rationnels $\{w \mid (X, w) \in P\}$.

Théorème 2.33 (Solutions rationnelles)**.** *Soit $G = (A, V, P)$ une grammaire étendue où $V = \{X_1, \ldots, X_n\}$. Il existe des langages rationnels $R_1, \ldots, R_n$ sur A tels que $(\overline{R_1}, \ldots, \overline{R_n})$ soit solution de $\overline{\mathcal{S}(G)}$.*

On commence par un lemme.

Lemme 2.34. *Soit K un langage sur $A + \{X\}$ et soit L et M deux langages sur A. Alors l'égalité $\overline{K(L^*M)L^*} = \overline{K(M)L^*}$ est vérifiée.*

Preuve. Soit u un mot de K. En regroupant les occurrences de la variable X à la fin, on obtient $\overline{u} = \overline{wX^n}$ où w est un mot sur A et où n est le nombre d'occurrences de X dans u. On vérifie alors facilement que l'égalité $\overline{u(L^*M)L^*} = \overline{u(M)L^*}$. On obtient le résultat du lemme en sommant les égalités précédentes sur tous les éléments de K. □

On procède maintenant à la preuve du théorème.

Preuve. La preuve du théorème se fait par récurrence sur le nombre n de variables de la grammaire étendue G.

Commençons par une grammaire $G = (A, \{X\}, P)$ ayant une seule variable X. Les règles de G sont notées $S \to S(X)$ où $S(X)$ est une expression rationnelle sur l'alphabet $(A + \{X\})^*$. Une solution du système $\overline{\mathcal{S}(G)}$ est un langage L tel que $\overline{L} = \overline{S(L)}$.

On isole dans $S(X)$ les mots ayant au moins une occurrence de X et les mots ayant aucune occurrence de X. On pose donc $P(X) = S(X) \cap (A + \{X\})^* X (A + \{X\})^*$ et $T = S(X) \cap A^*$. Les deux langages $P(X)$ (on note $P(X)$ plutôt que P pour insister sur le fait que les mots de P contiennent des occurrences de la variable X) et T sont deux langages rationnels puisque la classe des langages rationnels est close par intersection.

Comme tous les mots de $P(X)$ contiennent au moins une occurrence de X, on peut écrire modulo la commutativité

$$\overline{P(X)} = \overline{Q(X)X}$$

où $Q(X)$ peut être choisi rationnel. Il suffit de prendre pour $Q(X)$ l'ensemble des mots de $P(X)$ où la première occurrence de X a été supprimée.

Le système $\overline{\mathcal{S}(G)}$ s'écrit alors

$$\overline{L} = \overline{Q(L)L + T}.$$

On va vérifier que le langage rationnel $R = Q(T)^*T$ est effectivement une solution du système $\overline{\mathcal{S}(G)}$.

$$\begin{aligned}
\overline{Q(R)R + T} &= \overline{Q(Q(T)^*T)Q(T)^*T + T} \\
&= \overline{Q(T)Q(T)^*T + T} && \text{par le lemme précédent} \\
&= \overline{Q(T)^*T} && Q(T)^* = Q(T)Q(T)^* + \varepsilon \\
&= \overline{R}.
\end{aligned}$$

Ceci termine le cas où la grammaire a une seule variable.

Soit n un entier positif. On considère maintenant une grammaire étendue G ayant $n+1$ variables $X_0, X_1, \ldots, X_n$. Les règles de G s'écrivent

$$X_i \to S_i(X_0, \ldots, X_n) \quad \text{pour } 0 \leq i \leq n$$

où chaque S_i est un langage rationnel sur $A+X$ avec $X = \{X_0, \ldots, X_n\}$. Par commodité, on note X' l'ensemble X privé de la variable X_0, c'est-à-dire l'ensemble $\{X_1, \ldots, X_n\}$.

On introduit la grammaire G_0 obtenue en considérant les variables $X_1, \ldots, X_n$ comme des lettres terminales et en ne prenant que les règles associées à X_0 dans G. Plus formellement, la grammaire G_0 est donnée par

$$G_0 = (A + X', \{X_0\}, \{X_0 \to S_0(X_0, X')\}).$$

Comme la grammaire G_0 a une seule variable, on utilise le cas précédent pour obtenir une solution $R_0(X')$ du système $\overline{\mathcal{S}(G_0)}$ qui vérifie donc l'égalité suivante.

$$\overline{R_0(X')} = \overline{S_0(R_0(X'), X')} \tag{2.1}$$

On introduit la grammaire étendue G' en remplaçant, dans les autres règles, X_0 par la solution $R_0(X')$ de $\overline{\mathcal{S}(G_0)}$. Plus formellement, la grammaire G' est donnée par

$$G' = (A, X', \{X_i \to S_i(R_0(X'), X') \mid 1 \leq i \leq n\}).$$

Il faut remarquer que la grammaire G' est étendue même si la grammaire G ne l'est pas. En effet, la variable X_0 est substituée par un langage rationnel. Ceci explique la nécessité de considérer des grammaires étendues pour la preuve du théorème.

Puisque G' a n variables, l'hypothèse de récurrence peut être appliquée. Il existe n langages rationnels $R_1, \ldots, R_n$ sur A qui forment une solution $R' = (R_1, \ldots, R_n)$ du système $\overline{\mathcal{S}(G')}$. Ils vérifient donc les égalités

$$\overline{R_i} = \overline{S_i(R_0(R'), R')} \quad \text{pour } 1 \leq i \leq n.$$

Il reste à vérifier que le $n+1$-uplet $R = (R_0(R'), R_1, \ldots, R_n)$ constitue effectivement une solution du système $\overline{\mathcal{S}(G)}$ de la grammaire initiale G. En substituant X' par R' dans l'équation (2.1), on obtient une équation qui complète les n équations ci-dessus pour montrer que le $n+1$-uplet R est bien une solution rationnelle de $\overline{\mathcal{S}(G)}$. Ceci termine la preuve du théorème. □

Preuve du théorème de Parikh

Soit L un langage algébrique. Il existe une grammaire $G = (A, V, P)$ où $X = \{X_1, \ldots, X_n\}$ telle que $L = L_G(X_1)$. Si L est propre, on peut supposer G propre. D'après la proposition 2.31, $\overline{L_G}$ est l'unique solution propre du système $\overline{\mathcal{S}(G)}$. D'après le théorème 2.33, ce même système admet une solution rationnelle $R = (R_1, \ldots, R_n)$. En combinant les deux résultats, on conclut que $\overline{L} = \overline{R_1}$.

Si L n'est pas propre, on applique le résultat précédent au langage $L' = L \setminus \{\varepsilon\}$ qui est propre et aussi algébrique. On obtient un langage rationnel R' tel que $\overline{L'} = \overline{R'}$. Le langage $R = R' + \varepsilon$ vérifie alors l'égalité $\overline{L} = \overline{R}$ souhaitée.

2.3 Arbres de dérivation

Dans une dérivation, les règles qui s'appliquent à des variables différentes peuvent être appliquées de manière indépendante et donc dans un ordre quelconque. Ceci conduit à avoir plusieurs dérivations différentes même lorsqu'il y a essentiellement qu'une seule façon d'obtenir un mot. Les arbres de dérivation permettent de capturer cette notion d'ambiguïté des grammaires. Ils permettent aussi de formaliser un lemme d'itération.

Définition 2.35 (Arbre de dérivation)**.** Soit $G = (A, V, P)$ une grammaire. Un *arbre de dérivation* est un arbre fini étiqueté par $A \cup V \cup \{\varepsilon\}$ vérifiant la propriété suivante. Si S est l'étiquette d'un nœud interne et si $a_1, \ldots, a_n$ sont les étiquettes de ses fils alors $S \to a_1 \cdots a_n$ est une règle de G. La *frontière* d'un arbre de dérivation est le mot obtenu par concaténation des étiquettes des feuilles de gauche à droite. Si la frontière contient au moins une variable, l'arbre est un *arbre de dérivation partielle*

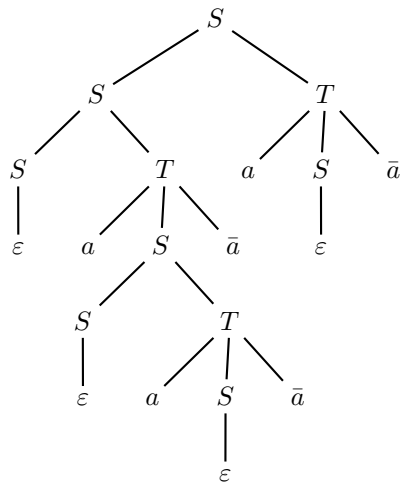

FIGURE 2.1 – Arbre de dérivation

Exemple 2.36. Soit la grammaire définie par les règles $\{S \to ST + \varepsilon, T \to aS\bar{a}\}$. La figure 2.1 exhibe un arbre de dérivation pour cette grammaire. La frontière de cet arbre est le mot $aa\bar{a}\bar{a}a\bar{a}$.

Proposition 2.37. *Le langage $L_G(S)$ (resp. $\widehat{L_G}(S)$) est l'ensemble des mots $w \in A^*$ (resp. $(A+V)^*$) tels qu'il existe un arbre de dérivation (resp. partielle) ayant S à la racine et dont la frontière est w.*

Preuve. La preuve est immédiate dans la mesure où un arbre de dérivation est simplement une autre façon de visualiser une dérivation. Une preuve formelle peut être faite par récurrence sur la longueur de la dérivation. □

2.3.1 Ambiguïté

La notion d'ambiguïté est l'analogue pour les grammaires du non déterminisme des automates. De manière intuitive, une grammaire est non ambiguë, si elle engendre chaque mot d'une façon au plus. Contrairement au cas des automates, toute grammaire n'est pas équivalente à une grammaire non ambiguë. Certains langages algébriques ne sont engendrés que par des grammaires ambiguës.

Définition 2.38 (Grammaire ambiguë)**.** Une grammaire G est dite *ambiguë* s'il existe un mot ayant au moins deux arbres de dérivation distincts étiqueté à la racine par la même variable S.

La définition suivante est justifiée par l'existence de langages algébriques uniquement engendrés par des grammaires ambiguës.

Définition 2.39 (Langage ambigu)**.** Un langage algébrique est dit *non ambigu* s'il existe au moins une grammaire non ambiguë qui l'engendre. Sinon, il est dit *inhéremment ambigu* pour signifier que toute grammaire qui l'engendre est ambiguë.

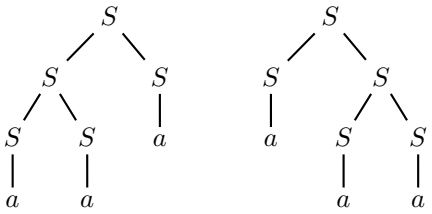

FIGURE 2.2 – Deux arbres de dérivation pour aaa

Exemple 2.40. La grammaire $S \to SS + a$ est ambiguë car le mot aaa a deux arbres de dérivations (cf. figure 2.2). Par contre, le langage engendré $L_G(S) = a^+$ est non ambigu car il est également engendré par la grammaire $S \to aS + a$ qui est non ambiguë.

Les propositions 2.47 et 2.49 établissent qu'il existe des langages algébriques inhéremment ambigus.

2.3.2 Lemme d'itération

Le but de cette partie est d'établir un analogue du lemme de l'étoile pour les langages algébriques. Pour cette raison, ce lemme dû à Ogden est aussi appelé *lemme d'itération*. Il permet de montrer que certains langages ne sont pas algébriques ou que certains

langages algébriques sont inhéremment ambigus. On commence par un lemme purement combinatoire sur les arbres.

Soit un arbre où certaines feuilles sont *distinguées*. On dit que :
- un nœud est *distingué* lorsque le sous-arbre dont il est racine contient des feuilles distinguées.
- un nœud est *spécial* lorsqu'il a au moins deux fils distingués.

Par définition, un nœud spécial est aussi distingué. Le parent d'un nœud distingué est encore distingué. En particulier, la racine de l'arbre est distinguée dès que l'arbre a au moins une feuille distinguée. Rappelons qu'un arbre est dit de *degré* m si chaque nœud a au plus m fils. Le lemme suivant établit un lien entre le nombre de feuilles distinguées et le nombre de nœuds spéciaux qui se trouvent sur une même branche.

Lemme 2.41. *Soit t un arbre de degré m avec k feuilles distinguées. Si chaque branche de t contient au plus r nœuds spéciaux, alors $k \leq m^r$.*

Preuve. Pour deux nœuds x et x' d'un arbre, on appelle *plus petit ancêtre commun* le nœuds z où les deux chemins de x et de x' à la racine se rejoignent. Les nœuds x et x' sont deux descendants de z mais ils ne sont pas descendants d'un même fils de z. Si x et x' sont deux feuilles distinguées, leur plus petit ancêtre commun est donc un nœud spécial.

On montre le résultat par récurrence sur r. Si $r = 0$, il n'y a aucun nœud spécial dans l'arbre. Il y a donc au plus une seule feuille distinguée. S'il y avait deux feuilles distinguées, leur plus petit ancêtre commun serait spécial.

On suppose maintenant que $r \geq 1$. Soit x un nœud spécial n'ayant pas de descendant spécial autre que lui-même. En appliquant le résultat pour $r = 0$ aux sous-arbres enracinés en les fils de x, on obtient que chacun de ces sous-arbres contient au plus une seule feuille distinguée. Comme x a au plus m fils, le sous-arbre enraciné en x a au plus m feuilles distinguées. On considère l'arbre obtenu en remplaçant chaque sous-arbre enraciné en un nœud spécial sans descendant spécial par une seule feuille distinguée. Le nombre de nœuds spéciaux sur chaque branche a diminué d'une unité. Par hypothèse de récurrence, l'arbre obtenu a au plus m^{r-1} feuilles distinguées. Comme chacune de ces nouvelles feuilles a remplacé un sous-arbre avec au plus m feuilles distinguées dans l'arbre initial, on obtient le résultat. □

Le résultat suivant est traditionnellement appelé lemme d'Ogden. Ce lemme n'est pas exempt de défaut. Il n'est pas facile à énoncer, la preuve est loin d'être triviale et surtout, il est assez délicat à utiliser. Il faut cependant s'en contenter car il est presque le seul outil dont on dispose pour montrer qu'un langage n'est pas algébrique.

Lemme 2.42 (d'Ogden)**.** *Pour toute grammaire $G = (A, V, P)$ et toute variable $S \in V$, il existe un entier K tel que tout mot $f \in \widehat{L_G}(S)$ ayant au moins K lettres distinguées se factorise en $f = \alpha u \beta v \gamma$, où $\alpha, u, \beta, v, \gamma \in (A + V)^*$, avec :*

1. *$S \xrightarrow{*} \alpha T \gamma$ et $T \xrightarrow{*} uTv + \beta$.*
2. *soit α, u et β, soit β, v et γ contiennent des lettres distinguées.*
3. *$u\beta v$ contient moins de K lettres distinguées.*

Les conditions du lemme montrent que les tous les mots de la forme $\alpha u^n \beta v^n \gamma$ pour $n \geq 0$ appartiennent au langage $\widehat{L_G}(S)$. Un telle paire (u, v) est appelée une *paire itérante*.

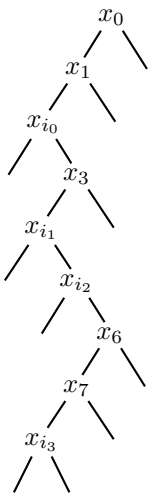

FIGURE 2.3 – La branche B avec $i_0 = 2$, $i_1 = 4$, $i_2 = 5$ et $i_3 = 8$

Preuve. Soit m la longueur maximale des membres droits des règles. Puisque f appartient à $\widehat{L_G}(S)$, il existe un arbre de dérivation t dont la racine est étiquetée par S et dont la frontière est le mot f. Par définition de m, l'arbre t est de degré m. Les feuilles distinguées de t sont celles étiquetées par les lettres distinguées de f.

On pose $r = 2|V| + 2$ et $K = m^r + 1$. Comme l'arbre t a K feuilles distinguées, il a, d'après le lemme précédent, au moins une branche B ayant au moins $r + 1$ nœuds spéciaux $x_0, \ldots, x_r$. Chaque nœud x_i a par définition au moins deux fils distingués dont l'un est sur la branche B. Le nœud x_i est dit *gauche* (respectivement *droit*) s'il a un autre fils distingué à gauche (respectivement à droite) de la branche B. Il peut être simultanément gauche et droit : il est alors arbitrairement considéré comme gauche. Parmi les $r + 1 = 2|V| + 3$ nœuds $x_0, \ldots, x_r$, au moins $|V| + 2$ d'entre eux sont tous gauches ou tous droits. On pose alors $k = |V| + 1$. Par symétrie, on suppose qu'il existe des indices $0 \leq i_0 < \cdots < i_k \leq r$ tels que les nœuds $x_{i_0}, \ldots, x_{i_k}$ soient gauches (cf. figure 2.3 pour un exemple). Comme k est supérieur au nombre de variables, deux nœuds x_{j_1} et x_{j_2} parmi les nœuds $x_{i_1}, \ldots x_{i_k}$ sont nécessairement étiquetés par la même variable T.

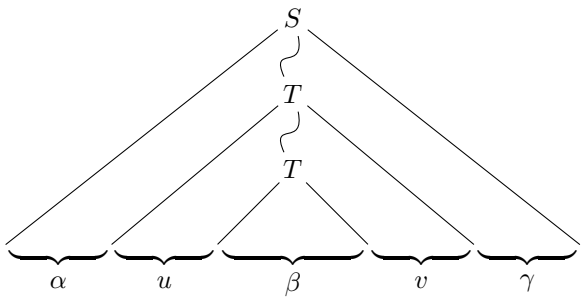

FIGURE 2.4 – Découpage d'un mot par le lemme d'Ogden

Le mot est alors découpé comme le suggère la figure 2.4. Le fait que le nœud x_{i_0} soit gauche garantit que α contienne des lettres distinguées. De même, les nœuds x_{j_1} et x_{j_2} garantissent respectivement que u et β contiennent des lettres distinguées. Si $u\beta v$ contient plus de K lettres distinguées, on recommence le raisonnement avec le sous-arbre de dérivation engendrant ce mot. □

Dans le cas où toutes les lettres de f sont marquées, on obtient le corollaire suivant.

Corollaire 2.43 (Théorème de Bar-Hillel, Perles et Shamir)**.** *Pour tout langage algébrique L, il existe $N \geq 0$ tel que pour tout mot $f \in L$, si $|f| \geq N$ alors on peut trouver une factorisation $f = \alpha u \beta v \gamma$ tel que $|uv| > 0$, $|u\beta v| < N$ et $\alpha u^n \beta v^n \gamma \in L$ pour tout $n \geq 0$.*

2.3.3 Applications du lemme d'itération

Dans cette partie, on applique le lemme d'itération pour montrer que certains langages classiques ne sont pas algébriques et que certains langages algébriques sont inhéremment ambigus. Nous commençons par une application directe du corollaire précédent.

Proposition 2.44. *Le langage $L = \{a^n b^n c^n \mid n \geq 0\}$ n'est pas algébrique.*

Preuve. Le corollaire précédent suffit pour prouver que le langage L n'est pas algébrique. Soit N l'entier fourni par le corollaire. On considère le mot $f_1 = a^N b^N c^N \in L$ qui se factorise $f_1 = \alpha u \beta v \gamma$ tel que $f_n = \alpha u^n \beta v^n \gamma \in L$ pour tout $n \geq 0$. Chacun des mots u et v ne contient qu'une seule des lettres a, b ou c. Sinon le mot f_2 n'appartient pas à $a^*b^*c^* \supset L$. Il en découle que f_2 ne peut pas appartenir à L car les nombres d'occurrences de a, b et c ne sont plus égaux. □

La proposition précédente a pour corollaire immédiat.

Corollaire 2.45. *La classe des langages algébriques n'est close ni par intersection ni par complémentation.*

Preuve. Le langage $\{a^n b^n c^n \mid n \geq 0\}$ est l'intersection des deux langages algébriques $\{a^m b^m c^n \mid m, n \geq 0\}$ et $\{a^m b^n c^n \mid m, n \geq 0\}$. Puisque la classe des langages algébriques est close par l'union, elle ne peut pas être close par complémentation. Sinon, elle serait aussi close par intersection.

On peut aussi montrer directement que la classe des langages algébriques n'est pas close par complémentation en considérant le langage $L = \{a^m b^n c^p \mid m \neq n \text{ ou } n \neq p\}$. Ce langage est bien sûr algébrique mais son complémentaire ne l'est pas. Pour montrer ce dernier fait, on constate que le langage $a^*b^*c^* \cap (A^* \setminus L)$ est égal à $\{a^n b^n c^n \mid n \geq 0\}$ qui n'est pas algébrique. On utilise la propriété que l'intersection d'un langages rationnel et d'un langage algébrique est encore algébrique (cf. proposition 2.53) pour conclure. □

Nous donnons maintenant une application du lemme d'itération où la possibilité de distinguer certaines lettres est cruciale. Il est important que le lemme s'applique aussi aux dérivations partielles.

Proposition 2.46. *Le langage $L = \{a^m b^n c^m d^n \mid n, m \geq 0\}$ n'est pas algébrique.*

Preuve. Supposons qu'il existe une grammaire G telle que $L = L_G(S)$. Soit k l'entier fourni par le lemme d'itération. On applique le résultat au mot $f = a^k b^k c^k d^k$ où les k lettres distinguées sont les lettres b. D'après le lemme d'itération, il existe des dérivations

$$S \xrightarrow{*} a^k b^{k_1} T d^{k_2} \qquad T \xrightarrow{*} b^i T d^i \qquad T \xrightarrow{*} b^{k'_1} c^k d^{k'_2}$$

avec les relations $k = k_1 + i + k'_1 = k_2 + i + k'_2$. On applique à nouveau le lemme d'itération au mot $a^k b^{k_1} T d^{k_2}$ où les k lettres distinguées sont les lettres a. On obtient immédiatement une contradiction car la paire itérante obtenue contient des lettres a mais aucune lettre d. □

Nous terminons par une application assez astucieuse du lemme d'itération pour montrer qu'il existe des langages inhéremment ambigus.

Proposition 2.47. *Le langage* $L = \{a^m b^m c^n \mid m, n \geq 0\} \cup \{a^m b^n c^n \mid m, n \geq 0\}$ *est inhéremment ambigu.*

Preuve. Chacun des langages $\{a^m b^m c^n \mid m, n \geq 0\}$ et $\{a^m b^n c^n \mid m, n \geq 0\}$ est bien sûr algébrique et le langage L est donc aussi algébrique. Soit $G = (A, V, P)$ une grammaire telle que $L = L_G(S_0)$. On montre que la grammaire G est ambiguë en montrant qu'un mot de la forme $a^n b^n c^n$ a au moins deux arbres de dérivation.

Soit k l'entier fourni par le lemme d'itération. On applique le résultat au mot $f_1 = a^k b^k c^{k+k!}$ où les k lettres distinguées sont les b. D'après le lemme d'itération, il existe des dérivations

$$S_0 \xrightarrow{*} \alpha S \gamma \xrightarrow{*} \alpha u S v \gamma \xrightarrow{*} \alpha u \beta v \gamma = f_1$$

Les conditions impliquent que $u = a^i$ et $v = b^i$ pour un entier $0 \leq i \leq k$. En itérant $k!/i$ fois la dérivations $S \xrightarrow{*} uSv$ on obtient un arbre de dérivation pour le mot $f = a^{k+k!} b^{k+k!} c^{k+k!}$. Cet arbre contient un sous-arbre dont la frontière ne contient que des lettres a et b dont au moins $k! - k$ lettres b.

En appliquant le même procédé au mot $f_2 = a^{k+k!} b^k c^k$, on obtient un autre arbre de dérivation pour le même mot f. Cet arbre contient un sous-arbre dont la frontière ne contient que des lettres b et c dont au moins $k! - k$ lettres b. Cet arbre est donc différent du premier arbre trouvé. □

Exercice 2.48. Soit A un alphabet et $\#$ une nouvelle lettre n'appartenant pas à A. Montrer que les trois langages

$$L_1 = \{ww \mid w \in A^*\}$$
$$L_2 = \{w\#w \mid w \in A^*\}$$
$$L_3 = \{w\#w' \mid w, w' \in A^*, w \neq w' \text{ et } |w| = |w'|\}$$

ne sont pas algébriques.

Solution. On donne uniquement la solution pour les langages L_2 et L_3. D'après la proposition 2.53, il suffit de montrer que le langage $L'_2 = L_2 \cap a^+ b^+ \# a^+ b^+$ n'est pas algébrique. Ce langage est égal à $\{a^m b^n \# a^m b^n \mid m, n \geq 1\}$. Supposons que ce langage soit engendré par une grammaire G. Grâce au théorème de Bar-Hillel, Perles et Shamir, on obtient un entier N. On considère alors le mot $f = a^N b^N \# a^N b^N$. On trouve une factorisation

de f en $\alpha u\beta v\gamma$ tel que $|uv| > 0$, $|u\beta v| < N$. Cette dernière condition empêche d'avoir $u = v = a^p$ ou $u = v = b^q$. Le mot $\alpha u^2\beta v^2\gamma$ n'appartient pas L et c'est une contradiction.

Le langages L_3 est plus délicat à traiter. On considère à nouveau la langage $L_3' = L_3 \cap a^+b^+\#a^+b^+$ qui est égal à $\{a^mb^n\#a^pb^q \mid m+n = p+q \text{ et } m \neq p\}$. Si ce langage est algébrique, le lemme d'Ogden fournit un entier N et on note $M = N!$. On considère le mot $f = a^Nb^{N+2M}\#a^{N+M}b^{N+M}$. On applique le lemme d'Ogden en marquant les N premiers a. On obtient une factorisation $f = \alpha u\beta v\gamma$ tel que $|uv| > 0$, $|u\beta v| < N$. Le mot u est nécessairement de la forme a^k pour $k \leq N$. Le mot v est aussi de longueur k. Il est égal à a^k ou à b^k et ces lettres se situent à droite du $\#$. Si v est égal à b^k, on peut en pompant obtenir le mot $a^Nb^{N+2M}\#a^Nb^{N+2M}$ car k divise M. Comme ce mot n'est pas dans L, le mot v est égal à a^k. Le mot β est alors de la forme $a^ib^{N+2M}\#a^j$. On applique à nouveau le lemme d'Ogden au mot β en marquant N lettres b. On obtient une factorisation $\alpha'u'\beta'v'\gamma'$ où u' est de la forme $b^{k'}$ et v' est de longueur k' et ne contient que des a ou que des b. Si v' est de la forme $b^{k'}$, on obtient une contradiction en pompant. Si v' est de la forme $a^{k'}$, on recommence avec β'. Comme le nombre de b est beaucoup plus grand que le nombre de a, on aboutit à une contradiction après un certain nombre d'étapes.

2.3.4 Ambiguïté inhérente

Il est souvent difficile de montrer qu'un langage algébrique donné est inhéremment ambigu. Lorsque le lemme d'itération n'est pas adapté, une autre méthode consiste à utiliser la fonction génératrice du langage L définie par

$$f_L(z) = \sum_{n\geq 0} a_n z^n \qquad \text{où} \quad a_n = |L \cap A^n|.$$

Si le langage est non ambigu, la fonction $f_L(z)$ est algébrique car elle est solution du système commutatif associé à une grammaire non ambiguë qui engendre L. En utilisant cette méthode, on peut par exemple montrer le théorème suivant (cf. exemple 2.8 p. 77 pour la définition du langage de Goldstine).

Proposition 2.49 (Flageolet)**.** *Le langage de Goldstine est inhéremment ambigu.*

Preuve. Nous donnons juste une idée de la preuve qui fait appel à des éléments d'analyse pour lesquels nous renvoyons le lecteur à des ouvrages spécialisés. Un mot n'est pas dans le langage de Goldstine soit parce qu'il se termine par la lettre a soit parce qu'il est de la forme $baba^2ba^3b\cdots a^nb$ pour $n \geq 0$. Ceci montre que la fonction génératrice du langage de Goldstine est égale à $(1-z)/(1-2z) - g(z)$ où la fonction $g(z)$ est égale à

$$g(z) = z + z^3 + z^6 + \cdots = \sum_{n\geq 1} z^{n(n+1)/2}.$$

Une fonction de la forme $\sum_{n\geq 0} z^{c_n}$ où la suite $(c_n)_{n\geq 0}$ vérifie $\sup(c_{n+1} - c_n) = \infty$ converge sur le bord de son disque de convergence. La fonction g admet donc une infinité de singularités et ne peut être algébrique comme f_L. □

2.4 Propriétés de clôture

Cette partie est consacrée à quelques propriétés de clôture classiques des langages algébriques. Nous montrons que la classe des langages algébriques est close pour les opérations rationnelles, les substitutions algébriques, les morphismes inverses et l'intersection avec un langage rationnel. Ces différentes propriétés permettent d'aborder le théorème de Chomsky et Schützenberger qui établit que les langages de Dyck sont en quelque sorte les langages algébriques génériques. Tout langage algébrique est l'image par un morphisme d'une intersection d'un langage de Dyck avec un langage rationnel.

2.4.1 Opérations rationnelles

On commence par les opérations rationnelles qui montrent que les langages algébriques généralisent les langages rationnels.

Proposition 2.50 (Opérations rationnelles)**.** *Les langages algébriques sont clos par union, concaténation et étoile.*

Preuve. Soient $G = (A, V, P)$ et $G' = (A, V', P')$ deux grammaires telles que $L = L_G(S)$ et $L' = L_{G'}(S')$. On suppose sans perte de généralité que $V \cap V' = \varnothing$.

- Le langage $L + L'$ est égal à $L_{G_1}(S_0)$ où la grammaire G_1 est égale à $(A, \{S_0\} \cup V \cup V', \{S_0 \to S + S'\} \cup P \cup P')$
- Le langage LL' est égal à $L_{G_2}(S_0)$ où la grammaire G_2 est égale à $(A, \{S_0\} \cup V \cup V', \{S_0 \to SS'\} \cup P \cup P')$
- Le langage L^* est égal à $L_{G_3}(S_0)$ où la grammaire G_3 est égale à $(A, \{S_0\} \cup V, \{S_0 \to SS_0 + \varepsilon\} \cup P)$

□

Le corollaire suivant découle directement de la proposition précédente.

Corollaire 2.51. *Les langages rationnels sont aussi algébriques.*

Pour obtenir directement le corollaire, on peut également construire une grammaire qui engendre le langage $L(\mathcal{A})$ des mots acceptés par un automate normalisé $\mathcal{A} = (Q, A, E, \{i\}, \{f\})$. La grammaire $G = (A, Q, P)$ où l'ensemble des règles est $P = \{p \to aq \mid p \xrightarrow{a} q \in E\} \cup \{f \to \varepsilon\}$ vérifie que $L_G(i) = L(\mathcal{A})$. Réciproquement, une grammaire dont toutes les règles sont de la forme $S \to aT$ ou de la forme $S \to a$ pour $a \in A$ et $S, T \in V$ engendre un langage rationnel.

2.4.2 Substitution algébrique

On rappelle qu'une substitution de A^* dans B^* est un morphisme de A^* dans $\mathfrak{P}(B^*)$. Une substitution σ est dite *algébrique* si $\sigma(a)$ est un langage algébrique pour toute lettre a de A. Les morphismes sont bien sûr un cas particulier de substitution algébrique. L'image par un morphisme d'un langage algébrique est encore un langage algébrique.

Proposition 2.52 (Substitution algébrique)**.** *Si $L \subseteq A^*$ est un langage algébrique et σ une substitution algébrique de A^* dans B^*, alors le langage $\sigma(L) = \bigcup_{w \in L} \sigma(w)$ est aussi algébrique.*

Preuve. Soient $G = (A, V, P)$ une grammaire telle que $L = L_G(S)$ et $\sigma : A \to \mathfrak{P}(B^*)$, une substitution algébrique. Pour tout $a \in A$, soit $G_a = (B, V_a, P_a)$ une grammaire telle que $\sigma(a) = L_{G_a}(S_a)$. Quitte à renommer des variables, on peut supposer que les ensembles de variables de toutes ces grammaires sont disjoints deux à deux.

On considère alors la grammaire $G' = (B, V \cup \bigcup_a V_a, P' \cup \bigcup_a P_a)$ où $P' = \{S \to \rho(w) \mid S \to w \in P\}$ où le morphisme ρ est défini par $\rho(S) = S$ pour tout $S \in V$, et $\rho(a) = S_a$ pour tout $a \in A$. Cette grammaire engendre $\sigma(L)$. □

2.4.3 Intersection avec un rationnel

On montre dans cette partie que l'intersection d'un langage rationnel et d'un langage algébrique est encore un langage algébrique.

Proposition 2.53 (Intersection avec un rationnel)**.** *Si L est un langage algébrique et K un langage rationnel, alors $K \cap L$ est algébrique.*

L'intersection de deux langages algébriques n'est pas algébrique en général comme cela a été vu au corollaire 2.45 (p. 94).

Preuve. Soit $G = (A, V, P)$ une grammaire telle que $L = L_G(T)$. Sans perte de généralité, on peut supposer que G est en forme normale quadratique. Cette condition n'est pas indispensable à la preuve mais simplifie les notations.

Méthode des automates Soit $\mathcal{A} = (Q, A, E, \{i\}, \{f\})$ un automate normalisé acceptant le langage K. Le langage $K \cap L$ est engendré par la grammaire $G' = (A, V', P')$ définie de la façon suivante.

$$\begin{aligned} V' &= \{S_{pq} \mid S \in V \text{ et } p, q \in Q\}, \\ P' &= \{S_{pq} \to a \mid S \to a \in P \text{ et } p \xrightarrow{a} q \in E\}, \\ &\quad \cup \{S_{pq} \to R_{pr}T_{rq} \mid r \in Q \text{ et } S \to RT \in P\}. \end{aligned}$$

Le langage $K \cap L$ algébrique car il est égal à $L_{G'}(T_{if})$.

Méthode des monoïdes Soit $\mu : A^* \to M$ un morphisme de A^* dans un monoïde M fini tel que $K = \mu^{-1}(H)$ où H est une partie de M. Le langage $K \cap L$ est engendré par la grammaire $G' = (A, V', P')$ définie de la façon suivante.

$$\begin{aligned} V' &= \{S_m \mid S \in V \text{ et } m \in M\}, \\ P' &= \{S_m \to a \mid S \to a \in P \text{ et } m = \mu(a)\}, \\ &\quad \cup \{S_m \to R_{m_1}T_{m_2} \mid S \to RT \in P \text{ et } m = m_1 m_2\}. \end{aligned}$$

Le langage $K \cap L$ est algébrique car il est égal à l'union $\bigcup_{m \in H} L_{G'}(T_m)$.

□

2.4.4 Morphisme inverse

On montre dans cette partie que l'image inverse par un morphisme d'un langage algébrique est encore algébrique. Le résultat n'est pas immédiat pour la raison suivante. Tout mot w d'un langage algébrique a une factorisation $w = w_1 \cdots w_m$ induite par un arbre

de dérivation. Si le mot w est aussi l'image par un morphisme μ d'un mot $u = a_1 \cdots a_n$, il a une autre factorisation $w = w'_1 \cdots w'_n$ où $w'_i = \mu(a_i)$. Ces deux factorisations ne coïncident pas *a priori*. La preuve du résultat s'appuie sur une décomposition astucieuse d'un morphisme inverse avec des morphisme alphabétiques. Cette décomposition est en soi un résultat intéressant. On commence par la définition d'un morphisme alphabétique.

Définition 2.54 (Morphisme alphabétique)**.** Un morphisme $\sigma : A^* \to B^*$ est dit *alphabétique* (resp. *strictement alphabétique*) si pour tout $a \in A$, $|\sigma(a)| \leq 1$ (resp. $|\sigma(a)| = 1$).

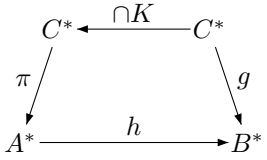

FIGURE 2.5 – Factorisation d'un morphisme

Lemme 2.55 (Factorisation d'un morphisme)**.** *Pour tout morphisme $h : A^* \to B^*$, il existe deux morphismes alphabétiques $g : C^* \to B^*$ et $\pi : C^* \to A^*$ et un langage rationnel $K \subseteq C^*$ tel que $h^{-1}(w) = \pi(g^{-1}(w) \cap K)$ pour tout mot $w \in B^*$.*

Preuve. On définit l'alphabet C et les deux parties C_0 et C_1 de C par

$$C = \{(a, i) \mid a \in A \text{ et } 0 \leq i \leq |h(a)|\}$$
$$C_0 = \{(a, 0) \mid a \in A\} \qquad C_1 = \{(a, |h(a)|) \mid a \in A\}$$

puis les langages rationnels locaux W et K de C^* par

$$W = C^2 \setminus (\{(a, i)(a, i + 1) \mid a \in A \text{ et } 0 \leq i < |h(a)|\} \cup C_1C_0)$$
$$K = (C_0C^* \cap C^*C_1) \setminus C^*WC^*$$

Les mots de K sont les mots qui se factorisent $u_1 \cdots u_n$ où chaque mot u_i pour $1 \leq i \leq n$ est un mot de la forme $(a_i, 0)(a_i, 1) \cdots (a_i, |h(a_i)|)$ pour une lettre a_i de A. L'appartenance d'un mot w au langage K est entièrement déterminée par la première et la dernière lettre de w ainsi que les paires de lettres consécutives dans w. Un langage ayant cette propriété est dit *local*.

On définit finalement les deux morphismes $g : C^* \to B^*$ et $\pi : C^* \to A^*$ par

$$g(a, i) = \begin{cases} \varepsilon & \text{si } i = 0 \\ h(a)[i] & \text{si } i \geq 1 \end{cases} \qquad \text{et} \qquad \pi(a, i) = \begin{cases} a & \text{si } i = 0 \\ \varepsilon & \text{si } i \geq 1 \end{cases}$$

où $h(a)[i]$ est une notation pour la i-ième lettre du mot $h(a)$. Il reste encore à vérifier que les morphismes définis conviennent mais c'est de la pure routine. □

Proposition 2.56 (Morphisme inverse)**.** *Si $h : A^* \to B^*$ est un morphisme et $L \subseteq B^*$ est un langage algébrique, alors $h^{-1}(L)$ est aussi algébrique.*

Preuve. Montrons d'abord la propriété lorsque h est alphabétique. Soit $G = (B, V, P)$ une grammaire et S_0 une variable telles que $L = L_G(S_0)$.

Puisque le morphisme h est alphabétique, l'alphabet A se partitionne en les deux parties A_0 et A_1 définies par $A_0 = \{a \in A \mid h(a) = \varepsilon\}$ et $A_1 = \{a \in A \mid h(a) \in B\}$. Le morphisme h est étendu à $(A + V)^*$ en posant $h(S) = S$ pour toute variable $S \in V$.

On définit la grammaire $G' = (A, V, P_0 \cup P_1)$ où les ensembles de règles P_0 et P_1 sont respectivement donnés par

$$P_0 = \{T \to \sum_{a \in A_0} aT + \varepsilon\},$$
$$P_1 = \{S \to Tu_1T \cdots Tu_nT \mid u_i \in (V \cup A_1)^*, S \to h(u_1 \cdots u_n) \in P\}.$$

On vérifie que la grammaire G' engendre le langage $h^{-1}(L)$. Ensuite, pour étendre ce résultat à un morphisme h quelconque, il suffit d'appliquer le lemme précédent. L'intersection avec le rationnel K découle de la propriété précédente. □

2.4.5 Théorème de Chomsky et Schützenberger

Le théorème suivant établit que les langages de Dyck (cf. exemple 2.8 p. 77 pour la définition) sont des langages algébriques *génériques*. Ils sont générateurs du *cône* des langages algébriques. On peut consulter [Aut87] pour une preuve plus détaillée de ce théorème.

Théorème 2.57 (Chomsky et Schützenberger)**.** *Un langage L est algébrique si et seulement si $L = \varphi(D_n^* \cap K)$ pour un entier n, un langage rationnel K et un certain morphisme φ alphabétique.*

Preuve. La condition est suffisante grâce aux propriétés de clôture des langages algébriques.

Réciproquement, soit $G = (A, V, P)$ une grammaire telle que $L = L_G(S_0)$ pour $S_0 \in V$. D'après la proposition 2.21, on peut supposer que G en forme normale quadratique. Les règles de G sont donc de la forme $S \to S_1S_2$ pour $S_1, S_2 \in V$ ou de la forme $S \to a$ pour $a \in A$. Pour chaque règle r du premier type, on introduit six nouveaux symboles a_r, b_r, c_r, $\bar{a}_r$, $\bar{b}_r$ et $\bar{c}_r$. Pour chaque règle r du second type, on introduit deux nouveaux symboles a_r et $\bar{a}_r$. Soit A' l'ensemble de toutes les nouvelles lettres introduites :

$$A' = \{a_r, b_r, c_r, \bar{a}_r, \bar{b}_r, \bar{c}_r \mid r = S \to S_1S_2\} \cup \{a_r, \bar{a}_r \mid r = S \to a\}.$$

Soit D_n le langage de Dyck sur l'alphabet de parenthèses A'. L'entier n est donc égal à trois fois le nombre de règles de la forme $S \to S_1S_2$ plus le nombre de règles de la forme $S \to a$. On définit alors la grammaire $G' = (A', V, P')$ où les règles de G' sont en correspondance avec les règles de G. Pour chaque règle r de G, il existe une règle r' de G' définie de la manière suivante.

- Si r est la règle $S \to S_1S_2$, alors r' est la règle $S \to a_rb_rS_1\bar{b}_rc_rS_2\bar{c}_r\bar{a}_r$.
- Si r est la règle $S \to a$, alors r' est la règle $S \to a_r\bar{a}_r$.

On définit finalement le morphisme $\varphi : A'^* \to A^*$ de la façon suivante. Si r est une règle de la forme $S \to S_1S_2$, on pose φ par $\varphi(a_r) = \varphi(b_r) = \varphi(c_r) = \varphi(\bar{a}_r) = \varphi(\bar{b}_r) = \varphi(\bar{c}_r) = \varepsilon$. Si r est une règle de la forme $S \to a$, on pose $\varphi(a_r) = a$ et $\varphi(\bar{a}_r) = \varepsilon$.

Les définitions des règles de G' et du morphisme φ impliquent immédiatement que $L = L_G(S_0) = \varphi(L_{G'}(S_0))$. L'inclusion $L_{G'}(S_0) \subseteq D_n^*$ découle directement de la forme particulière des règles de G'. Il reste maintenant à définir un langage rationnel K tel que $L_{G'}(S_0) = D_n^* \cap K$.

On va définir un langage local K qui exprime les contraintes entre les paires de lettres consécutives de A' dans $L_{G'}(S_0)$. Soit A'_0 l'ensemble des lettres a_r où r est une règle ayant S_0 pour membre gauche. Soit K le langage local $K = A'_0 A'^* \setminus A'^* W A'^*$ où l'ensemble W est défini par

$$\begin{aligned} A'^2 \setminus W = {} & \{a_r b_r, \bar{b}_r c_r, \bar{c}_r \bar{a}_r \mid r = S \to S_1 S_2\} \\ & \cup \{a_r \bar{a}_r \mid r = S \to a\} \\ & \cup \{b_r a_t, \bar{a}_t \bar{b}_r \mid r = S \to S_1 S_2 \text{ et } t = S_1 \to \ldots\} \\ & \cup \{c_r a_t, \bar{a}_t \bar{c}_r \mid r = S \to S_1 S_2 \text{ et } t = S_2 \to \ldots\}. \end{aligned}$$

L'inclusion $L_{G'}(S_0) \subset D_n^* \cap K$ découle directement du choix du langage K. L'inclusion inverse est un peu fastidieuse mais pas très difficile. Elle se montre par récurrence sur la longueur du mot. □

Pour un entier n, on note A_n l'alphabet $\{a_1, \ldots, a_n, \bar{a}_1, \ldots, \bar{a}_n\}$ ayant n paires de parenthèses.

Lemme 2.58. *Pour tout entier n, il existe un morphisme $\psi : A_n^* \to A_2^*$ tel que $D_n^* = \psi^{-1}(D_2^*)$.*

Preuve. Il est facile de vérifier que le morphisme ψ défini par $\psi(a_k) = a_1 a_2^k a_1$ et $\psi(\bar{a}_k) = \bar{a}_1 \bar{a}_2^k \bar{a}_1$ pour $1 \leq k \leq n$ convient parfaitement. □

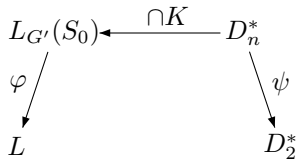

FIGURE 2.6 – Théorème de Chomsky et Schützenberger

D'après le lemme précédent, le théorème de Chomsky et Schützenberger établit que tout langage algébrique L s'écrit $L = \varphi(\psi^{-1}(D_2^*) \cap K)$ pour des morphismes φ et ψ et pour un langage rationnel K (cf. figure 2.6). Une transformation de la forme $X \mapsto \varphi(\psi^{-1}(X) \cap K)$ s'appelle une *transduction rationnelle*. Ces transformations sont en fait très naturelles. Elles peuvent en effet être réalisées de manière équivalente par des automates à deux bandes (une pour l'entrée et une pour la sortie) appelés transducteurs (cf. théorème 1.158 p. 72). Le décomposition d'un morphisme obtenue au lemme 2.55 montre qu'un morphisme inverse est une transduction rationnelle.

2.5 Forme normale de Greibach

La forme normale de Greibach est une forme particulière de grammaire. Toute grammaire est équivalente à une grammaire en forme normale de Greibach mais la preuve

n'est pas triviale, contrairement à la forme normale quadratique. Cette forme normale permet en particulier de montrer que les ε-transitions ne sont pas nécessaires dans les automates à pile.

Définition 2.59 (Forme normale de Greibach)**.** Une grammaire $G = (A, V, P)$ est en *forme normale de Greibach* si chacune de ses règles est de la forme $S \to w$ où w appartient à AV^*. Si de plus chaque w appartient à $A + AV + AV^2$, la grammaire est en *forme normale de Greibach quadratique*.

Les règles de la forme $S \to w$ où w appartient à AV ne peuvent pas être supprimées même si leur rôle se limite essentiellement à engendrer les mots de longueur 2.

Proposition 2.60 (Forme normale de Greibach)**.** *Toute grammaire propre est équivalente à une grammaire en forme normale de Greibach quadratique.*

La preuve initiale de Greibach donnait seulement une grammaire en forme normale non quadratique. Nous allons d'abord présenter une première preuve plus élémentaire qui donne une grammaire en forme normale de Greibach. Nous donnons ensuite la preuve due à Rosenkrantz qui est plus concise et donne en plus une grammaire en forme normale quadratique. Le théorème peut encore être raffiné en introduisant des formes normales bilatères où les membres droits des règles appartiennent à $A + AA + AVA + AVVA$. La preuve de ce résultat dû à Hotz est cependant plus difficile.

Preuve élémentaire. Il suffit de trouver une grammaire avec des règles de la forme $S \to w$ où $w \in A(A+V)^*$. On se ramène alors à une grammaire en forme normale de Greibach en introduisant une nouvelle variable T_a avec la règle $T_a \to a$ pour chaque lettre terminale a et en remplaçant chaque occurrence de a par T_a dans les autres règles.

Soit $G = (A, V, P)$ une grammaire où on suppose que $V = \{X_1, \ldots, X_n\}$. On pose $G_0 = G$ et on définit par récurrence une suite $G_0, \ldots, G_n$ de grammaires telles que dans chaque grammaire G_i les variables $X_1, \ldots, X_i$ n'apparaissent pas en tête des membres droits de règles. On suppose avoir construit la grammaire G_{i-1} et on construit la grammaire G_i par les deux étapes suivantes.

1. On commence par supprimer X_i en tête des productions de X_i. Soient

$$X_i \to X_i u_1 + \cdots + X_i u_k + w_1 + \cdots + w_p$$

les règles de G_{i-1} ayant X_i comme membre gauche où les mots $w_1, \ldots, w_p$ ne commencent pas par la variable X_i. Par hypothèse de récurrence, les mots $w_1, \ldots, w_p$ ne commencent pas par une variable X_j pour $1 \leq j \leq i-1$. On introduit une nouvelle variable X_i' et on remplace les règles précédentes par les nouvelles règles

$$X_i \to w_1 X_i' + \cdots + w_p X_i' + w_1 + \cdots + w_p$$
$$X_i' \to u_1 X_i' + \cdots + u_k X_i' + u_1 + \cdots + u_k.$$

2. La transformation précédente a pu faire apparaître des variables X_j (pour $1 \leq j \leq i$) en tête des mots u_l. En outre, la variable X_i peut être la première lettre de membres droits d'autres règles. Toutes ces occurrences des variables X_j (pour $1 \leq j \leq i$) en tête de membres droits sont supprimées en remplaçant chaque règle $S \to X_j v$ par toutes les règles $S \to uv$ où u parcourt les productions de la variable X_j.

□

Exemple 2.61. En appliquant l'algorithme de la preuve précédente à la grammaire $G = G_0$, on obtient successivement les grammaires G_1, G_2 et G_3.

$$G_0 \begin{cases} A \to AB + a \\ B \to BC + b \\ C \to CA + c \end{cases} \qquad G_1 \begin{cases} A \to aA' + a \\ A' \to BA' + B \\ B \to BC + b \\ C \to CA + c \end{cases}$$

$$G_2 \begin{cases} A \to aA' + a \\ A' \to bB'A' + bA' + bB' + b \\ B \to bB' + b \\ B' \to CB' + C \\ C \to CA + c \end{cases} \qquad G_3 \begin{cases} A \to aA' + a \\ A' \to bB'A' + bA' + bB' + b \\ B \to bB' + b \\ B' \to cC'B' + cB' + cC' + c \\ C \to cC' + c \\ C' \to aA'C' + aC' + aA' + a \end{cases}$$

Autre preuve (Rosenkrantz). Soit $G = (A, V, P)$ une grammaire où l'ensemble des variables est $V = \{X_1, \ldots, X_n\}$. Sans perte de généralité, on peut supposer que G est en forme normale quadratique. Cette hypothèse est uniquement nécessaire pour que la grammaire en forme normale de Greibach qui est obtenue soit aussi quadratique. Pour $1 \leq i \leq n$, on définit le langage P_i par $P_i = \{w \mid X_i \to w\}$. Les règles de G s'écrivent alors $X_i \to P_i$ pour $1 \leq i \leq n$. On introduit le vecteur $\vec{X} = (X_1, \ldots, X_n)$ et le vecteur $\vec{P} = (P_1, \ldots, P_n)$ de sorte que les règles s'écrivent

$$\vec{X} \to \vec{P}.$$

On introduit la matrice $R = (R_{ij})$ de dimensions $n \times n$ où chaque entrée R_{ij} est définie par $R_{ij} = X_i^{-1} P_j$. On définit aussi le vecteur $\vec{S} = (S_1, \ldots, S_n)$ où $S_i = P_i \cap AV^*$. Les règles de la grammaire s'écrivent alors sous forme matricielle de la manière suivante

$$\vec{X} \to \vec{X}R + \vec{S}.$$

Par analogie avec le système $X = XR + S$ dont la solution est SR^*, on introduit la matrice $Y = (Y_{ij})$ où chaque entrée Y_{ij} est une nouvelle variable. On introduit aussi la grammaire G' dont les règles sont données par

$$X \to SY$$
$$Y \to RY + I$$

où I est la matrice ayant ε sur la diagonale et $\varnothing$ partout ailleurs. La grammaire G' est équivalente à la grammaire G. Comme G est en forme normale quadratique, chaque entrée R_{ij} est une somme de variables de V. En utilisant $X \to SY$, on définit finalement la matrice $R' = (R'_{ij})$ où chaque entrée R'_{ij} est donnée par la formule

$$R'_{ij} = \sum_{X_k \in R_{ij}} (SY)_k.$$

La grammaire G'' dont les règles sont données par

$$X \to SY$$
$$Y \to R'Y + I$$

est encore équivalente à la grammaire G. On supprime les ε-règles de G'' pour obtenir une grammaire en forme normale de Greibach quadratique équivalente à G. □

2.6 Automates à pile

Les automates à pile sont une extension des automates finis. Outre un contrôle par un nombre fini d'états, ils possèdent une mémoire auxiliaire. Celle-ci est organisée sous la forme d'une pile contenant des symboles. Il est seulement possible d'empiler ou de dépiler des symboles. Seul le symbole en sommet de la pile est visible du contrôle. Les transitions effectuées ne dépendent que de l'état interne, du symbole au sommet de la pile et aussi du mot lu. Les langages acceptés par ces automates sont exactement les langages algébriques.

2.6.1 Définitions et exemples

Comme un automate fini, un automate à pile a un ensemble fini Q d'états. Il y a un alphabet A sur lequel sont écrits les mots d'entrée ainsi qu'un alphabet de pile qui contient tous les symboles qui peuvent être mis dans la pile. La transition effectuée par l'automate dépend de l'état de contrôle, de la lettre lue dans le mot d'entrée et du symbole au sommet de la pile. Chaque transition remplace le symbole au sommet de la pile par un mot éventuellement vide sur l'alphabet de pile. Ce dernier cas correspond à un dépilement. Effectuer une transition fait passer à la lettre suivante du mot d'entrée qui est donc lu séquentiellement de gauche à droite. On introduit aussi des ε-transitions qui ne lisent aucune lettre du mot d'entrée. Ces transitions ne sont pas absolument nécessaires dans le cas des automates non déterministes. Il est possible de les supprimer même si ce n'est pas immédiat. Par contre, ces ε-transitions sont indispensables pour les automates déterministes.

Définition 2.62 (Automate à pile)**.** Un *automate à pile* est constitué d'un alphabet d'entrée A, d'un alphabet de pile Z dont un symbole initial $z_0 \in Z$, d'un ensemble fini d'états Q dont un état initial q_0 et de transitions de la forme $q, z \xrightarrow{y} q', h$ avec $q, q' \in Q$, $y \in A \cup \{\varepsilon\}$, $z \in Z$ et $h \in Z^*$. Le mot y qui est soit une lettre soit le mot vide est l'*étiquette* de la transition.

L'état initial q_0 est l'état de contrôle dans lequel se trouve l'automate à pile au début d'un calcul. Tout au long d'un calcul, la pile de l'automate n'est jamais vide. Le calcul se bloque dès que la pile devient vide. Le symbole de pile initial z_0 est le symbole qui est mis dans la pile avant de commencer tout calcul.

Une configuration d'un automate est un état instantané de l'automate qui comprend l'état de contrôle et le contenu de la pile. Le contenu de la pile est vu comme un mot sur l'alphabet de pile. On appelle *configuration* une paire (q, h) de $Q \times Z^*$. La configuration initiale est (q_0, z_0) où q_0 et z_0 sont respectivement l'état initial et le symbole de pile initial. La notion de calcul pour un automate à pile est l'équivalent d'un chemin pour un automate fini. Il est constitué d'une suite d'étapes de calcul consistant à effectuer une transition de l'automate.

Définition 2.63 (Calcul d'un automate à pile)**.** Une *étape de calcul* est une paire de configurations (C, C') notée $C \xrightarrow{y} C'$ telles que $C = (p, zw)$, $C' = (q, hw)$ et $p, z \xrightarrow{y} q, h$

est une transition de l'automate. Un *calcul* de l'automate est une suite d'étapes de calcul consécutives :

$$C_0 \xrightarrow{y_1} C_1 \xrightarrow{y_2} \cdots \xrightarrow{y_n} C_n.$$

Le mot $y_1 \cdots y_n$ est l'*étiquette* du calcul.

Effectuer une transition $p, z \xrightarrow{y} q, h$ d'un automate à pile consiste à passer de l'état p à l'état q, à lire le mot y et à remplacer le symbole z du sommet de pile par le mot h. Une transition est impossible dès que la pile est vide et le calcul se bloque. La transition peut soit dépiler si h est vide, soit remplacer le symbole de sommet de pile si h est de longueur 1, soit empiler si h est de longueur au moins 2. On parle d'ε-transition lorsque y est le mot vide.

On utilise la même notation pour une transition et une étape de calcul dans la mesure où une transition $p, z \xrightarrow{y} q, h$ peut être vue comme une étape de calcul de la configuration (p, z) à la configuration (q, h) en prenant $w = \varepsilon$.

Dans la définition précédente, le contenu de la pile est écrit de haut en bas. Le sommet de pile se situe donc à gauche du mot de pile. Certains ouvrages utilisent la convention inverse où le sommet de pile se situe à droite. Aucune des conventions n'est parfaite et n'évite de passer au mot miroir dans certaines preuves. La convention adoptée dans cet ouvrage simplifie la preuve d'équivalence entre grammaires et automates à pile (cf. théorème 2.68). Par contre, la preuve en complément que l'ensemble des contenus forme un langage rationnel (cf. théorème 2.81) nécessite un passage au mot miroir.

2.6.2 Différents modes d'acceptation

Il reste à définir les configurations finales qui déterminent les calculs acceptants et par conséquent les mots acceptés par l'automate. Il existe plusieurs modes d'acceptation pour définir ces configurations finales. Les principaux modes utilisés sont les suivants. Ils sont tous équivalents pour les automates non déterministes. Pour les automates déterministes, le mode d'acceptation par pile vide est plus faible car le langage accepté est nécessairement un langage préfixe.

pile vide : les configurations finales sont toutes les configurations de la forme (q, ε) où la pile est vide.

état final : les configurations finales sont toutes les configurations de la forme (q, w) où l'état q appartient à un sous-ensemble F d'états distingués de Q et où le contenu de la pile est quelconque.

sommet de pile : les configurations finales sont les configurations (q, zw) où le sommet de pile z appartient à un sous-ensemble Z_0 de symboles de pile distingués de Z.

combinaison : toute combinaison des trois premiers.

Un mot f est *accepté* par un automate à pile s'il existe un calcul d'étiquette f de la configuration initiale (q_0, z_0) à une configuration finale.

Exemple 2.64. Soit l'automate à pile défini sur les alphabets d'entrée et de pile $A = \{a, b\}$ et $Z = \{z\}$ avec les états $Q = \{q_0, q_1\}$ et comportant les trois transitions suivantes :

$$q_0, z \xrightarrow{a} q_0, zzz \qquad q_0, z \xrightarrow{\varepsilon} q_1, \varepsilon \qquad q_1, z \xrightarrow{b} q_1, \varepsilon.$$

Un calcul valide pour cet automate est par exemple

$$q_0, z \xrightarrow{a} q_0, zzz \xrightarrow{\varepsilon} q_1, zz \xrightarrow{b} q_1, z \xrightarrow{b} q_1, \varepsilon.$$

Si on choisit un arrêt par pile vide, cet automate accepte le langage $L_1 = \{a^n b^{2n} \mid n \geq 0\}$. Si on choisit l'arrêt par état final avec l'ensemble $F = \{q_1\}$, il accepte le langage $L_2 = \{a^n b^p \mid 0 \leq p \leq 2n\}$.

Exemple 2.65. Soit l'automate à pile défini sur les alphabets d'entrée et de pile $A = \{a, b\}$ et $Z = \{A, B\}$ avec $z_0 = A$ comme symbole de pile initial. Il possède les états $Q = \{q_0, q_1, q_2\}$ et comporte les transitions suivantes :

$$\begin{array}{lll}
q_0, A \xrightarrow{a} q_1, A & & q_2, A \xrightarrow{a} q_2, \varepsilon \\
q_0, A \xrightarrow{b} q_1, B & & q_2, B \xrightarrow{b} q_2, \varepsilon \\
q_1, A \xrightarrow{a} q_1, AA & q_1, A \xrightarrow{b} q_2, BA & q_1, A \xrightarrow{a} q_2, A \\
q_1, B \xrightarrow{a} q_1, AB & q_1, B \xrightarrow{b} q_2, BB & q_1, B \xrightarrow{a} q_2, B \\
q_1, A \xrightarrow{b} q_1, BA & q_1, A \xrightarrow{a} q_2, AA & q_1, A \xrightarrow{b} q_2, A \\
q_1, B \xrightarrow{b} q_1, BB & q_1, B \xrightarrow{a} q_2, AB & q_1, B \xrightarrow{b} q_2, B
\end{array}$$

Cet automate à pile accepte par pile vide l'ensemble des palindromes non vides (cf. exercice 2.9 p. 78). Il fonctionne de la façon suivante. Dans la première moitié du mot, l'automate empile les lettres lues. Dans la seconde moitié du mot, il vérifie que les lettres lues coïncident avec les lettres dépilées. Plus précisément, les deux transitions de q_0 à q_1 remplacent le symbole de pile initial par la lettre lue. Tant que l'automate est dans l'état q_1, il empile la lettre lue. Le passage de q_1 à q_2 se fait soit en empilant la lettre lue pour un palindrome de longueur paire soit en l'ignorant pour un palindrome de longueur impaire. Dans l'état q_2, l'automate dépile et vérifie que les lettres coïncident. Cet automate n'a aucun moyen de déterminer la moitié du mot. Il est donc essentiel qu'il soit non déterministe.

Proposition 2.66 (Équivalence des modes d'acceptation)**.** *Les différents modes d'acceptation sont équivalents dans la mesure où ils permettent tous d'accepter exactement les mêmes langages (les langages algébriques).*

Un des problème des automates à pile est que le calcul se bloque dès que la pile devient vide. Cette difficulté peut être contournée en utilisant des automates à fond de pile testable qui peuvent tester si le sommet de pile est le dernier symbole dans la pile. Un automate est dit *à fond de pile testable* si son alphabet de pile est partitionné $Z = Z_0 \uplus Z_1$ de sorte que le contenu de la pile de toute configuration accessible a un symbole de Z_0 au fond de la pile et des symboles de Z_1 au dessus. Ceci signifie que le contenu de pile appartient à $Z_1^* Z_0$. Lorsque l'automate voit un sommet de pile de Z_0 il sait que celui-ci est le dernier symbole de la pile.

Il est possible de transformer un automate à pile quelconque en un automate à fond de pile testable. Pour cela, il faut doubler la taille de l'alphabet de pile en introduisant une copie $\bar{z}$ de chaque lettre z de Z. On pose alors $Z_0 = \{\bar{z} \mid z \in Z\}$ et $Z_1 = Z$. Chaque transition $p, z \xrightarrow{y} q, h$ donne alors les deux transitions $p, z \xrightarrow{y} q, h$ et $p, \bar{z} \xrightarrow{y} q, h'\bar{z}'$ si

$h = h'z'$ avec $h' \in Z^*$ et $z' \in Z$. La première transition est utilisée lorsque la pile contient au moins deux symboles et la seconde lorsque $\bar{z}$ est l'unique symbole dans la pile.

Preuve. Nous montrons uniquement l'équivalence entre les acceptations par pile vide et par état d'acceptation. Les autres preuves sont similaires.

Soit un automate qui accepte par pile vide. On suppose qu'il est à fond de pile testable avec une partition $Z = Z_0 \uplus Z_1$ des symboles de pile. On ajoute un nouvel état q_+ qui devient l'unique état final. Ensuite, chaque transition $q, z \xrightarrow{y} q', \varepsilon$ avec $z \in Z_0$ qui vide la pile est remplacée par une transition $q, z \xrightarrow{y} q_+, \varepsilon$ qui fait passer dans l'état final.

Soit un automate qui accepte par état d'acceptation. On suppose encore qu'il est à fond de pile testable avec une partition $Z = Z_0 \uplus Z_1$ des symboles de pile. Il faut faire attention que certains calculs peuvent se bloquer en vidant la pile. On ajoute deux nouveaux états q_- et q_+ et toutes les transitions de la forme $q_+, z \xrightarrow{\varepsilon} q_+, \varepsilon$ pour $z \in Z$. Ces transitions permettent de vider la pile dès que l'état q_+ est atteint. Pour chaque transition $q, z \xrightarrow{y} q', h$ où q' est final, on ajoute une transition $q, z \xrightarrow{y} q_+, \varepsilon$ permettant d'atteindre l'état q_+. Toute transition $q, z \xrightarrow{y} q', \varepsilon$ avec $z \in Z_0$ qui vide la pile sans atteindre un état final est remplacée par une transition $q, z \xrightarrow{y} q_-, z$ qui fait passer dans q_- sans vider la pile. □

Exercice 2.67. Donner un automate à pile qui accepte le langage de Dyck D_n^* (cf. exemple 2.8 p. 77 pour la définition).

Solution. On construit un automate à pile $\mathcal{A}$ qui accepte le langage de Dyck par état d'acceptation. L'automate $\mathcal{A}$ a deux états q_0 et q_1. L'état q_0 est initial et l'état q_1 est final. L'alphabet de pile est $Z = \{z_0, \bar{a}_1, \ldots, \bar{a}_n\}$ où z_0 est le symbole de pile initial. L'ensemble E des transitions est donné ci-dessous.

$$\begin{aligned} E = &\{q_0, z \xrightarrow{a_i} q_0, \bar{a}_i z \mid 1 \leq i \leq n \text{ et } z \in Z\}, \\ &\cup \{q_0, \bar{a}_i \xrightarrow{\bar{a}_i} q_0, \varepsilon \mid 1 \leq i \leq n\}, \\ &\cup \{q_0, z_0 \xrightarrow{\varepsilon} q_1, \varepsilon\}. \end{aligned}$$

Cet automate fonctionne de la façon suivante. Si l'automate lit a_i, la lettre $\bar{a}_i$ est empilé quelque soit le symbole de haut de pile. Si l'automate lit $\bar{a}_i$, le symbole de haut de pile doit être $\bar{a}_i$ et celui-ci est dépilé. Cet automate accepte aussi le langage de Dyck par pile vide.

On appelle *automate à un compteur* un automate à pile à fond de pile testable avec seulement deux symboles de pile. La partition de l'alphabet de pile est nécessairement $Z_0 = \{z_0\}$ et $Z_1 = \{z_1\}$. Si Z_0 ou Z_1 est vide, le nombre de contenus de pile est fini et l'automate à pile est en fait un automate fini. Les contenus de la pile sont les mots de $z_1^* z_0$. La seule information pertinente de la pile est le nombre de symboles dans celle-ci. La pile est en fait utilisée comme un compteur pouvant contenir un entier positif ou nul. Ceci justifie la terminologie. Le fond de pile testable permet à l'automate de tester si la valeur du compteur est nulle. L'automate construit dans la solution de l'exercice 2.67 est un automate à un compteur pour $n = 1$. Pour $n \geq 2$, le langage de Dyck D_n^* ne peut pas être accepté par un automate à un compteur.

2.6.3 Équivalence avec les grammaires

On montre dans cette partie que les automates à pile sont équivalents aux grammaires dans le sens où les langages acceptés par les automates à pile sont exactement les langages engendrés par les grammaires.

Théorème 2.68 (Équivalence grammaires/automates à pile)**.** *Un langage $L \subseteq A^*$ est algébrique si et seulement si il existe un automate à pile qui accepte L.*

Preuve. On commence par montrer que pour toute grammaire G, il existe un automate à pile qui accepte les mots engendrés par G. Soit $G = (A, V, P)$ une grammaire telle que $L = L_G(S_0)$. On suppose que toute règle de G est soit de la forme $S \to w$ avec $w \in V^*$, soit de la forme $S \to a$ avec $a \in A$. Il suffit pour cela d'introduire une nouvelle variable V_a pour chaque lettre a de A. Cette transformation est identique à la première étape de la mise en forme normale quadratique (cf. la preuve de la proposition 2.21).

On construit un automate à pile sur l'alphabet d'entrée A. Son alphabet de pile est l'ensemble V des variables de G et le symbole de pile initial est la variable S_0. cet automate a un unique état q_0 et il accepte par pile vide. Il possède une transition pour chaque règle de G. Son ensemble de transitions est donné par

$$\{q_0, S \xrightarrow{a} q_0, \varepsilon \mid S \to a \in P\} \cup \{q_0, S \xrightarrow{\varepsilon} q_0, h \mid S \to h \in P\}.$$

Pour les règles du premier ensemble, une lettre de l'entrée est lue et un symbole de pile est dépilé. Pour les règles du second ensemble, aucune lettre de l'entrée n'est lue et un ou plusieurs symboles de piles sont empilés. Il est facile de vérifier que cet automate simule les dérivations gauches de la grammaire.

Lorsque la grammaire est en forme normale de Greibach, l'automate équivalent peut être construit sans utiliser d'ε-transition. On suppose que toute règle de S est de la forme $S \to aw$ où $a \in A$ et $w \in V^*$. L'ensemble des transitions de l'automate est alors donné par

$$\{q_0, S \xrightarrow{a} q_0, w \mid S \to aw \in P\}.$$

Pour la réciproque, on utilise la méthode des triplets de Ginsburg. Soit un langage L accepté par un automate à pile $\mathcal{A}$ acceptant par pile vide. Pour simplifier les notations, on suppose que chaque transition de l'automate empile au plus deux symboles. Si cette hypothèse n'est pas vérifiée, il suffit de décomposer les transitions qui empilent plus de symboles en plusieurs transitions. Il faut alors ajouter quelques états intermédiaires. On note q_0 et z_0 l'état initial et le symbole de pile initial de l'automate $\mathcal{A}$.

Soient q et q' deux états et z un symbole de pile de l'automate. On note $L_{q,q',z}$ l'ensemble des mots f tels qu'il existe un calcul d'étiquette f de la configuration (q, z) à la configuration (q', ε). Ceci signifie que l'automate peut partir de l'état q avec juste le symbole z dans la pile, lire entièrement le mot f et arriver dans l'état q' avec la pile vide. Si un tel calcul est possible, il y a aussi un calcul d'étiquette f de la configuration (q, zw) à la configuration (q', w), pour tout mot w sur l'alphabet de pile. On a alors les

égalités suivantes.

$$
\begin{aligned}
L &= \bigcup_{q' \in Q} L_{q_0,q',z_0} \\
L_{q,q',z} &= \{y \mid q, z \xrightarrow{y} q', \varepsilon\} \\
&\cup \bigcup_{q,z \xrightarrow{y} q_1,z_1} y L_{q_1,q',z_1} \\
&\cup \bigcup_{q,z \xrightarrow{y} q_1,z_1 z_2} y L_{q_1,q_2,z_1} L_{q_2,q',z_2}
\end{aligned}
$$

La première égalité traduit simplement que l'automate $\mathcal{A}$ accepte le langage L par pile vide et que l'état initial et le symbole de pile initial sont respectivement q_0 et z_0. La seconde égalité s'obtient en analysant le début du calcul de (q, z) à (q', ε). Si la première transition dépile z, le calcul s'arrête immédiatement et le mot f est égal à l'étiquette y de la transition. Si la première transition remplace z par z_1, il faut alors un calcul qui dépile z_1. Si la première transition remplace z par $z_1 z_2$, il y a alors un calcul qui dépile d'abord z_1 puis z_2. L'évolution de la hauteur de pile lors de ce calcul est illustrée à la figure 2.7.

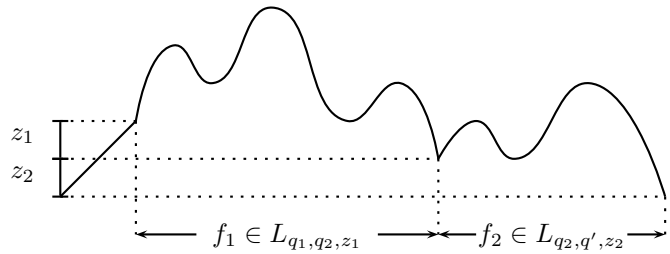

FIGURE 2.7 – Évolution de la hauteur de pile

Ces égalités se traduisent directement en une grammaire ayant pour variables les triplets de la forme (q, q', z). Il est à remarquer que si l'automate ne possède pas d'ε-transition, alors la grammaire obtenue est en forme normale de Greibach. □

Il faut remarquer que la traduction d'une grammaire en un automate donne un automate ayant un seul état. Ceci montre que tout automate à pile est en fait équivalent à un automate à pile ayant un seul état. Si de plus la grammaire est en forme normale de Greibach, l'automate n'a pas d'ε-transition. Ceci montre que tout automate à pile est équivalent à un automate à pile sans ε-transition.

Exemple 2.69. La construction d'un automate à partir de la grammaire $\{S \to aSa + bSb + a + b + \varepsilon\}$ pour engendrer les palindromes donne l'automate suivant. Cet automate a comme alphabet de pile $Z = \{S, A, B\}$ avec $z_0 = S$ comme symbole de pile initial. Cet

automate a un seul état q et ses transitions sont les suivantes.

$$q, S \xrightarrow{\varepsilon} q, ASA \qquad q, S \xrightarrow{\varepsilon} q, A \qquad q, S \xrightarrow{\varepsilon} q, \varepsilon \qquad q, A \xrightarrow{a} q, \varepsilon$$
$$q, S \xrightarrow{\varepsilon} q, BSB \qquad q, S \xrightarrow{\varepsilon} q, B \qquad\qquad\qquad q, B \xrightarrow{b} q, \varepsilon$$

Cet automate est à comparer avec l'automate de l'exemple 2.65.

Exemple 2.70. La méthode des triplets appliquée à l'automate de l'exemple 2.64, donne la grammaire $\{R \to aRRR, S \to aRRS + aRST + aSTT + \varepsilon, T \to b\}$ où les variables R, S, et T correspondent aux triplets (q_0, q_0, z), (q_0, q_1, z) et (q_1, q_1, z). Après réduction et substitution de T par b, on obtient la grammaire $\{S \to aSbb + \varepsilon\}$ qui engendre bien le langage $L_1 = \{a^n b^{2n} \mid n \geq 0\}$.

L'exercice suivant établit l'équivalence entre les grammaires linéaires et les automates à pic. La méthode pour convertir un automate à pic en une grammaire linéaire peut être étendue à tous les automates pour obtenir une autre preuve que les automates à pile acceptent des langages algébriques.

Exercice 2.71. Une grammaire $G = (A, V, P)$ est dite *linéaire* si chacune de ses règles $S \to w$ vérifie $w \in A^*VA^* + A^*$. Cela signifie qu'au plus une seule variable apparaît dans chaque membre droit de règle. Un langage L est dit *linéaire* s'il est engendré par une grammaire linéaire, c'est-à-dire si L est égal à $L_G(S)$ pour une variable S d'une grammaire linéaire G.

1. Montrer que l'image $\mu(L)$ par un morphisme $\mu : A^* \to B^*$ d'un langage linéaire L est encore un langage linéaire.
2. Montrer que l'intersection d'un langage linéaire et d'un langage rationnel est un langage linéaire.

Pour une configuration $C = (q, h)$ d'un automate à pile, on note $|C|$ le nombre $|h|$ de symboles dans la pile. Un automate à pile $\mathcal{A}$ est dit *à pic* si pour tout calcul $C_0 \xrightarrow{y_1} C_1 \xrightarrow{y_2} \cdots \xrightarrow{y_n} C_n$, il existe un entier $0 \leq k < n$ tel pour tout entier $0 \leq i < n$, si $i < k$, alors $|C_i| \leq |C_{i+1}|$ et si $i \geq k$, alors $|C_i| \geq |C_{i+1}|$. Ceci signifie que la hauteur de la pile croît au sens large de C_0 à C_k puis décroît de C_k à C_n.

3. Montrer qu'un langage linéaire est accepté par un automate à pic.

On s'intéresse maintenant à la réciproque.

4. Montrer que tout automate à pile $\mathcal{A}$ est équivalent à un automate à pile $\mathcal{A}'$ dont chacune des transitions $p, z \xrightarrow{y} q, h$ vérifie $|h| \leq 2$. Montrer en outre que si l'automate $\mathcal{A}$ est à pic, alors l'automate $\mathcal{A}'$ peut aussi être choisi à pic.
5. Montrer que tout automate à pic $\mathcal{A}$ est équivalent à un automate à pic $\mathcal{A}'$ dont chacune des transitions $p, z \xrightarrow{y} q, h$ vérifie $h = \varepsilon$, $h = z$ ou $h \in Zz$ où Z est l'alphabet de pile de $\mathcal{A}'$.

Soit $\mathcal{A}$ un automate à pic. On introduit une nouvelle lettre a_τ pour chaque transition τ de $\mathcal{A}$. On note $\mathcal{A}'$ l'automate obtenu en remplaçant chaque transition $\tau = q, z \xrightarrow{y} q', h$ par la transition $q, z \xrightarrow{a_\tau} q', h$. Les étiquettes des transitions de $\mathcal{A}'$ sont alors des lettres distinctes.

6. Montrer que si le langage accepté par $\mathcal{A}'$ est linéaire, alors le langage accepté par $\mathcal{A}$ est linéaire.

Soit $\mathcal{A}'$ un automate à pic ayant les deux propriétés suivantes. Les étiquettes des transitions sont des lettres distinctes et chaque transition est de la forme $q, z \xrightarrow{a} q', h$ avec $h = \varepsilon$, $h = z$ ou $h \in Zz$ où Z est l'alphabet de pile de $\mathcal{A}$. On introduit un automate $\mathcal{A}_0$ sans pile et un automate à pic $\mathcal{A}_1$ avec deux états. Les états (respectivement initiaux et finaux) de $\mathcal{A}_0$ sont ceux de $\mathcal{A}$ et ses transitions sont les transitions de la forme $p \xrightarrow{a} q$ s'il existe z et h tels que $p, z \xrightarrow{a} q, h$ soit une transition de $\mathcal{A}$. L'automate $\mathcal{A}_1$ a un état q_0 et qui est initial et un état q_1 qui est final. Son alphabet de pile est celui de $\mathcal{A}$. Son ensemble E_1 de transitions est défini par la formule suivante.

$$\begin{aligned} E_1 = &\{q_0, z \xrightarrow{y} q_0, h \mid \exists p \in Q \;\; p, z \xrightarrow{y} q, h \in E \text{ et } |h| \geq 1\} \\ \cup\; &\{q_0, z \xrightarrow{y} q_1, h \mid \exists p \in Q \;\; p, z \xrightarrow{y} q, h \in E \text{ et } |h| = 0\} \\ \cup\; &\{q_1, z \xrightarrow{y} q_1, h \mid \exists p \in Q \;\; p, z \xrightarrow{y} q, h \in E \text{ et } |h| \leq 1\} \end{aligned}$$

7. Montrer que le langage accepté par $\mathcal{A}'$ est égal à l'intersection des langages acceptés par $\mathcal{A}_0$ et $\mathcal{A}_1$.
8. Montrer que le langage accepté par $\mathcal{A}_1$ est linéaire.
9. Montrer qu'un langage accepté par un automate à pic est linéaire.

Solution.

1. Soit $G = (A, V, P)$ une grammaire telle $L = L_G(S_0)$ pour une variable S_0. Le morphisme μ peut être étendu à $(A+V)^*$ en posant $\mu(S) = S$ pour toute variable S. Le langage $\mu(L)$ est égal à $L_{G'}(S_0)$ où la grammaire $G' = (A, V, P')$ est définie par $P' = \{S \to \mu(w) \mid S \to w \in P\}$. Si la grammaire G est linéaire, la grammaire G' est encore linéaire.

2. La grammaire construite dans la preuve de la proposition 2.53 est encore linéaire si la grammaire du langage algébrique est linéaire.

3. L'automate construit dans la preuve du théorème 2.68 pour le passage d'une grammaire à un automate à pile n'est pas un automate à pic même si la grammaire est linéaire. Soit $G = (A, V, P)$ une grammaire linéaire. On construit un automate à pile $\mathcal{A}$ acceptant par pile vide. Ses alphabets d'entrées et de pile sont respectivement A et $A + V$. Cet automate possède un état initial q_0. Pour chaque règle $S \to uTv$ où $u = a_1 \cdots a_n$, on introduit $n - 1$ nouveaux états $q_1, \ldots, q_{n-1}$ et les transitions suivantes :

$$\begin{aligned} &q_{i-1}, S \xrightarrow{a_i} q_i, S \qquad \text{pour } 1 \leq i \leq n-1 \\ &q_{n-1}, S \xrightarrow{a_n} q_0, Tv. \end{aligned}$$

   On ajoute finalement les transitions

$$q_0, a \xrightarrow{a} q_0, \varepsilon \qquad \text{pour } a \in A.$$

   Cet automate est à pic et il accepte $L_G(S_0)$ si le symbole de pile initial est S_0.

4. Soit $\mathcal{A}$ un automate à pile quelconque. Chaque transition $p, z \xrightarrow{y} q, h$ où $|h| > 2$ est remplacée par plusieurs transitions. Si $h = z_1 \cdots z_n$, on introduit $n - 2$ nouveaux

états $q_2, \ldots, q_{n-1}$ et la transition $p, z \xrightarrow{y} q, h$ est remplacée par les $n-1$ transitions

$$\begin{aligned} p, z &\xrightarrow{y} q_{n-1}, z_{n-1}z_n \\ q_i, z_i &\xrightarrow{\varepsilon} q_{i-1}, z_{i-1}z_i \qquad \text{pour } 3 \leq i \leq n-1 \\ q_2, z_2 &\xrightarrow{\varepsilon} q, z_1 z_2. \end{aligned}$$

Si l'automate de départ est à pic, l'automate obtenu est encore à pile. Chaque transition qui faisait croître la taille de la pile de $n-1$ unités a été remplacée par $n-1$ transitions qui font croître la pile d'une seule unité.

5. Cette construction est un peu plus délicate. L'idée principale est de construire un automate qui mémorise le sommet de pile dans l'état et qui le remplace par un autre symbole.
6. La langage accepté par l'automate $\mathcal{A}$ est l'image du langage accepté par $\mathcal{A}'$ par le morphisme qui envoie chaque lettre a_τ sur la lettre ou le mot vide qui étiquette la transition τ dans $\mathcal{A}$. D'après le résultat de la question 1, le langage accepté par $\mathcal{A}$ est linéaire.
7. Dans la mesure où les lettres qui étiquettent les transitions sont distinctes, le mot d'entrée détermine complètement les calculs dans $\mathcal{A}$, $\mathcal{A}_0$ et$\mathcal{A}_1$. Le fait que le calcul sur une entrée f soit réussi dans $\mathcal{A}_0$ et $\mathcal{A}_1$ signifie respectivement que la suite d'états et la suite des mouvements de piles sont corrects. Le calcul est donc aussi réussi dans $\mathcal{A}$. La réciproque est évidente.
8. On constate que l'automate $\mathcal{A}_1$ est à pic. Un calcul commence dans l'état q_0 où les transition augmentent la taille de la pile. Le calcul passe ensuite dans l'état q_1 où les transitions diminuent la taille de la pile. Chaque lettre de l'entrée lue par une transition qui empile un symbole S est en correspondance avec la lettre qui dépile le symbole S. On en déduit immédiatement une grammaire linéaire pour le langage accepté par $\mathcal{A}_1$.
9. Le langage accepté par $\mathcal{A}_0$ est rationnel et celui accepté par $\mathcal{A}_1$ est linéaire. D'après les résultats des questions 2 et 7, le langage accepté par $\mathcal{A}'$ est linéaire. D'après la question 6, le langage accepté par $\mathcal{A}$ est linéaire.

2.6.4 Automates à pile déterministes

La définition d'un automate à pile déterministe est technique même si l'idée intuitive est relativement simple. L'intuition est qu'à chaque étape de calcul, il n'y a qu'une seule transition possible. Ceci signifie deux choses. D'une part, si une ε-transition est possible, aucune autre transition n'est possible. D'autre part, pour chaque lettre de l'alphabet d'entrée, une seule transition au plus est possible.

Définition 2.72. Un automate à pile (Q, A, Z, E, q_0, z_0) est *déterministe* si pour toute paire (p, z) de $Q \times Z$,

- soit il existe une unique transition de la forme $p, z \xrightarrow{\varepsilon} q, h$ et il n'existe aucune transition de la forme $p, z \xrightarrow{a} q, h$ pour $a \in A$,
- soit il n'existe pas de transition de la forme $p, z \xrightarrow{\varepsilon} q, h$ et pour chaque lettre $a \in A$, il existe au plus une transition de la forme $p, z \xrightarrow{a} q, h$.

Exemple 2.73. L'automate de l'exemple 2.64 n'est pas déterministe. Pour la paire (q_0, z), cet automate possède une ε-transition $q_0, z \xrightarrow{\varepsilon} q_1, \varepsilon$ et une transition $q_0, z \xrightarrow{a} q_0, zzz$. Le langage $L_1 = \{a^n b^{2n} \mid n \geq 0\}$ est accepté par pile vide par l'automate à pile déterministe ayant les états $Q = \{q_0, q_1, q_2\}$ et comportant les quatre transitions suivantes :

$$q_0, z \xrightarrow{a} q_1, zz \qquad q_1, z \xrightarrow{a} q_1, zzz \qquad q_1, z \xrightarrow{b} q_2, \varepsilon \qquad q_2, z \xrightarrow{b} q_2, \varepsilon.$$

Dans le cas des automates à pile déterministes, les différents modes d'acceptation ne sont plus équivalents. L'acceptation par pile vide permet seulement d'accepter des langages préfixes, c'est-à-dire des langages tels que deux mots du langage ne sont jamais préfixe l'un de l'autre.

Un langage algébrique est dit *déterministe* s'il est accepté par un automate à pile déterministe. L'intérêt de cette classe de langages est sa clôture par complémentation qui montre par ailleurs que cette classe est une sous-classe stricte des langages algébriques (cf. corollaire 2.45 p. 94).

Proposition 2.74. *Le complémentaire d'un langage algébrique déterministe est un langage algébrique déterministe.*

Preuve. La preuve de ce théorème n'est pas très difficile mais elle recèle quelques difficultés techniques. Soit $\mathcal{A}$ un automate à pile déterministe acceptant par état final. Soient E et F son ensemble de transitions et son ensemble d'états finaux. La première difficulté est que certains calculs de $\mathcal{A}$ peuvent se bloquer avant d'avoir entièrement lu l'entrée. Ce blocage peut survenir soit parce que l'automate n'est pas complet soit parce que la pile devient vide. La seconde difficulté est qu'un calcul ayant lu entièrement l'entrée puisse encore être prolongé avec des ε-transitions. Il s'ensuit qu'il n'y a pas vraiment unicité du calcul bien que l'automate soit déterministe. Ce prolongement peut en outre être infini.

La première difficulté se résout facilement avec les techniques habituelles. On peut supposer que l'automate est à fond de pile testable. La procédure pour rendre un automate à fond de pile testable préserve en effet le déterminisme de l'automate. L'alphabet de pile Z est alors partitionné $Z = Z_0 \uplus Z_1$ et le contenu de pile de chaque configuration accessible appartient à $Z_1^* Z_0$.

Pour éviter les blocages de l'automate, on ajoute deux états q_+ et q_- ainsi que les transitions suivantes :

$$\{q_+, z \xrightarrow{a} q_-, z \mid a \in A \text{ et } z \in Z\} \cup \{q_-, z \xrightarrow{a} q_-, z \mid a \in A \text{ et } z \in Z\}.$$

On complète alors l'automate. Pour toute paire $(p, z) \in Q \times Z$ telle qu'il n'existe pas de transition de la forme $p, z \xrightarrow{y} q, h$, avec $y \in A \cup \{\varepsilon\}$, on ajoute une transition $p, z \xrightarrow{\varepsilon} q_-, z$. On supprime ensuite les transitions qui vident la pile. On remplace chaque transition $p, z \xrightarrow{y} q, \varepsilon$ où z appartient à Z_0 par la transition $p, z \xrightarrow{y} q_+, z$ si q est final ou par la $p, z \xrightarrow{y} q_-, z$ sinon. Si l'état q_+ est ajouté à l'ensemble des états finaux, l'automate ainsi transformé accepte les mêmes mots que $\mathcal{A}$.

Il faut remarquer qu'il est tout à fait possible que l'automate obtenu ne puisse pas lire entièrement une entrée f bien qu'il soit complet. Le calcul sur f peut, après avoir lu un préfixe de f, se poursuivre par une infinité d'ε-transitions et ne jamais lire la fin du mot f. Une des transformations que nous allons appliquer va supprimer ces calculs infinis uniquement constitués d'ε-transitions.

La seconde difficulté provient des ε-transitions qui peuvent prolonger un calcul après la lecture de la dernière lettre. Elle se résout par plusieurs transformations successives de l'automate. On commence par transformer l'automate pour qu'il vérifie la propriété suivante. Son ensemble d'états est partitionné en deux ensembles Q_0 et Q_1 et tout calcul se termine dans un état de Q_1 s'il a visité un état final après la lecture de la dernière lettre de A et il se termine dans un état de Q_0 sinon. La transformation de l'automate introduit une copie $\bar{q}$ de chaque état q. On pose alors $Q_0 = Q$ et $Q_1 = \{\bar{q} \mid q \in Q\}$. Les transitions de l'automate sont alors les suivantes :

$$\begin{aligned}
E' = &\{p, z \xrightarrow{a} q, h \mid p, z \xrightarrow{a} q, h \in E \text{ et } q \notin F\}\\
\cup\ &\{\bar{p}, z \xrightarrow{a} q, h \mid p, z \xrightarrow{a} q, h \in E \text{ et } q \notin F\}\\
\cup\ &\{p, z \xrightarrow{a} \bar{q}, h \mid p, z \xrightarrow{a} q, h \in E \text{ et } q \in F\}\\
\cup\ &\{\bar{p}, z \xrightarrow{a} \bar{q}, h \mid p, z \xrightarrow{a} q, h \in E \text{ et } q \in F\}\\
\cup\ &\{p, z \xrightarrow{\varepsilon} q, h \mid p, z \xrightarrow{\varepsilon} q, h \in E \text{ et } q \notin F\}\\
\cup\ &\{\bar{p}, z \xrightarrow{\varepsilon} \bar{q}, h \mid p, z \xrightarrow{\varepsilon} q, h \in E \text{ et } q \notin F\}\\
\cup\ &\{p, z \xrightarrow{\varepsilon} \bar{q}, h \mid p, z \xrightarrow{\varepsilon} q, h \in E \text{ et } q \in F\}.
\end{aligned}$$

Les transitions étiquetées par une lettre a passent dans un état de Q_0 ou de Q_1 suivant que l'état d'arrivée est final ou non. Les ε-transitions font passer dans Q_1 dès qu'un état final est rencontré et font rester dans Q_1 ensuite.

Nous sommes maintenant en mesure de traiter les suites d'ε-transitions qui peuvent prolonger un calcul après la lecture de la dernière lettre. Soit $C = (q, z)$ une configurations avec un seul symbole z dans la pile. On considère le calcul maximal formé d'ε-transitions qui peut être effectué à partir de C. Ce calcul peut être fini ou non. Si il est fini, il se termine soit dans une configuration avec une pile vide soit dans une configuration avec une pile non vide mais où aucune ε-transition ne peut être effectuée. La possibilité que la pile se vide n'est pas exclue car nous n'avons pas supposé que z appartienne à Z_0. Soit X l'ensemble des configurations (p, z) telles que le calcul maximal formé d'ε-transitions à partir de (p, z) soit infini. On note X_0 (resp. X_1) l'ensemble des configurations de X telle que ce calcul infini formé d'ε-transitions ne rencontre pas d'état final (resp. rencontre un état final), c'est-à-dire reste dans les états de Q_0 (resp. dans des états de Q_1). Nous appliquons une nouvelle transformation qui supprime les calculs infinis formés d'ε-transitions. Chaque transition $p, z \xrightarrow{\varepsilon} q, h$ où (p, z) appartient à X est remplacée par une transition $p, z \xrightarrow{\varepsilon} q_-, z$ si (p, z) appartient à X_0 et par une transition $p, z \xrightarrow{\varepsilon} q_+, z$ si (p, z) appartient à X_1.

Nous appliquons une dernière transformation qui permet de prendre en compte les calculs maximaux finis formés d'ε-transitions. On note Y l'ensemble des configurations (p, z) telles que p appartient à Q_0 et qu'il n'existe pas de transition de la forme $p, z \xrightarrow{\varepsilon} q, h$. Puisque l'automate est complet, il y a pour chaque (p, z) de Y et chaque lettre a une transition de la forme $p, z \xrightarrow{a} q_a, h_a$. On ajoute alors pour chaque configuration (p, z) un nouvel état p_z, une transition $p, z \xrightarrow{\varepsilon} p_z, z$ et des transitions $p_z, z \xrightarrow{a} q_a, h_a$ qui remplacent alors les transitions $p, z \xrightarrow{a} q_a, h_a$.

Il faut remarquer que les calculs formés d'ε-transitions à partir d'une configuration (p, zw) qui se terminent en vidant le z de la pile n'ont pas été modifiés. Ils se poursuivent à partir d'une nouvelle configuration $(p', z'w')$ où $w = z'w'$ et peuvent à nouveau vider

z' de la pile et ainsi de suite jusqu'à arriver finalement à une configuration sans d'ε-transition possible.

Soit $\mathcal{A}'$ l'automate obtenu à l'issue de toutes les transformations et soit $F' = \{q_-\} \cup \{p_z \mid (p, z) \in Y\}$ son ensemble d'états finaux. On vérifie que l'automate $\mathcal{A}'$ accepte le complémentaire du langage accepté par l'automate $\mathcal{A}$.

Les ensembles X, X_0, X_1 et Y peuvent être effectivement calculés à partir de $\mathcal{A}$ et par conséquent l'automate $\mathcal{A}'$ aussi. □

Une autre propriété importante des langages déterministe est la suivante.

Proposition 2.75. *Tout langage algébrique déterministe est non ambigu.*

Preuve. La construction par les triplets de Ginsburg construit une grammaire équivalente à un automate. Les dérivations de la grammaire sont alors en bijection avec les calculs de l'automate de départ. On obtient donc une grammaire non ambiguë si chaque mot accepté est l'étiquette d'un unique calcul.

Le problème est qu'il n'y pas unicité du calcul dans un automate à pile déterministe à cause des ε-transitions qui peuvent prolonger un calcul acceptant après la lecture de la dernière lettre. Par contre, on peut forcer l'unicité du calcul en utilisant des transformations similaires à celles utilisées dans la preuve de la proposition précédente. On obtient alors une grammaire non ambiguë en appliquant la méthode des triplets de Ginsburg. □

La réciproque de la proposition précédente est fausse comme le montre l'exemple suivant. Cet exemple montre également que la classe des langages déterministes n'est pas fermée pour l'union.

Exemple 2.76. Le langage $L = \{a^n b^n \mid n \geq 0\} \cup \{a^n b^{2n} \mid n \geq 0\}$ est non ambigu. Par contre, il n'est pas déterministe. Même si l'idée intuitive de ce fait est relativement claire, la preuve en est particulièrement technique.

2.7 Compléments

Cette partie est consacrée à quelques compléments sur le groupe libre et les contenus de pile des automates. Dans les deux cas, l'approche est basée sur des réécritures. On commence donc par rappeler quelques résultats très classiques de réécriture.

2.7.1 Réécriture

On rappelle dans cette partie quelques résultats sur la terminaison et la confluence des relations qui seront utilisés par la suite. Pour une relation binaire notée $\to$, on note $\xrightarrow{*}$ la clôture réflexive et transitive de la relation $\to$. On dit qu'un élément x *se réduit* en y si $x \xrightarrow{*} y$.

Un élément x est dit *irréductible* ou *en forme normale* s'il n'existe pas d'élément y tel que $x \to y$. Une relation est *noethérienne* s'il n'existe pas de suite infinie $(x_n)_{n\geq 0}$ telle que $x_0 \to x_1 \to x_2 \to \cdots$. L'intérêt des relations noethériennes est que tout élément se réduit en un élément en forme normale.

Proposition 2.77. *Si la relation $\to$ est noethérienne, tout élément se réduit en un élément en forme normale.*

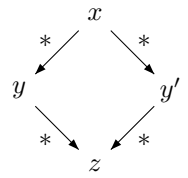

FIGURE 2.8 – Propriété de confluence

Une relation $\to$ est *confluente* si pour tous éléments x, y et y' tels que $x \xrightarrow{*} y$ et $x \xrightarrow{*} y'$, il existe un élément z tel que $y \xrightarrow{*} z$ et $y' \xrightarrow{*} z$. Cette propriété est représentée par la figure 2.8. Elle aussi appelée propriété diamant. La confluence est parfois appelée propriété de Church-Rosser par référence à la confluence de la β-réduction du λ-calcul qui a été prouvée par Church et Rosser. L'intérêt principal de la confluence est qu'elle garantit l'unicité de la forme normale.

Proposition 2.78. *Si la relation $\to$ est confluente, tout élément se réduit en au plus un élément en forme normale.*

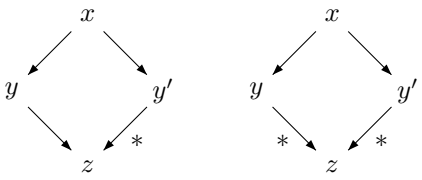

FIGURE 2.9 – Confluences forte et locale

La confluence est souvent difficile à vérifier. Pour contourner cette difficulté, on introduit deux propriétés plus faciles à vérifier. Une relation $\to$ est *fortement confluente* si pour tous éléments x, y et y' tels que $x \to y$ et $x \to y'$, il existe un élément z tel que $y = z$ ou $y \to z$ et $y' \xrightarrow{*} z$. Une relation $\to$ est *localement confluente* si pour tous éléments x, y et y' tels que $x \to y$ et $x \to y'$, il existe un élément z tel que $y \xrightarrow{*} z$ et $y' \xrightarrow{*} z$. Les propriétés de confluence forte et de confluence locale sont représentées à la figure 2.9. La terminologie est cohérente car la confluence forte implique bien la confluence.

Proposition 2.79. *Toute relation fortement confluente est confluente.*

Preuve. Soient trois éléments x, y et y' tels que $x \xrightarrow{*} y$ et $x \xrightarrow{*} y'$. Par définition, il existe deux suites $y_0, \ldots, y_m$ et $y'_0, \ldots, y'_n$ d'éléments telles que $y_0 = y'_0 = x$, $y_m = y$, $y'_n = y'$, $y_i \to y_{i+1}$ pour $0 \leq i \leq m-1$ et $y'_i \to y'_{i+1}$ pour $0 \leq i \leq n-1$. On montre l'existence de z tel que $y \xrightarrow{*} z$ et $y' \xrightarrow{*} z$ par récurrence sur n. Si $n = 0$, le résultat est trivial puisque $y' = x$ et que $z = y$ convient. On suppose maintenant que $n \geq 1$. On définit par récurrence une suite d'éléments $z_0, \ldots, z_m$ de la manière suivante. On pose $z_0 = y'_1$. On suppose avoir défini z_i tel que $y_i \to z_i$. On applique alors la confluente locale pour trouver z_{i+1} tel que $y_{i+1} \to z_{i+1}$ et $z_i \xrightarrow{*} z_{i+1}$. On peut alors appliquer l'hypothèse de récurrence à y'_1 puisque y'_1 se réduit en $n-1$ étapes à y' et en certain nombre d'étapes à z_m On trouve alors z tel que $z_m \xrightarrow{*} z$ et $y' \xrightarrow{*} z$ et donc tel que $y \xrightarrow{*} z$ et $y' \xrightarrow{*} z$. □

FIGURE 2.10 – Relation localement confluente mais non confluente

La confluence locale n'implique pas la confluence dans le cas général. La relation représentée à la figure 2.10 est localement confluente mais elle n'est pas confluente. Par contre, elle n'est pas noethérienne puisque qu'on a la suite de réductions $x \to x' \to x \to \cdots$. Le résultat suivant est souvent appelé lemme de Newman.

Proposition 2.80 (Newman)**.** *Toute relation noethérienne et localement confluente est confluente.*

La confluence d'une relation de réécriture est souvent une façon d'obtenir une forme normale unique. Dans ce cas, la réécriture doit aussi être noethérienne pour assurer l'existence de la forme normale. L'hypothèse du lemme précédent est alors naturelle et peu contraignante.

Preuve. Soit une relation $\to$ noethérienne et localement confluente. Puisque $\to$ est noethérienne, tout élément se réduit à au moins un élément en forme normale. On va montrer que tout élément se réduit en fait à un seul élément en forme normale. Cela suffit pour prouver la confluence de la relation $\to$. Si les trois éléments x, y, et y' vérifient $x \xrightarrow{*} y$ et $x \xrightarrow{*} y'$, il existe des éléments z et z' en forme normale tels que $y \xrightarrow{*} z$ et $y' \xrightarrow{*} z'$. On déduit des relations $x \xrightarrow{*} z$ et $z \xrightarrow{*} z'$ que $z = z'$.

Supposons qu'il existe deux éléments en forme normale z et z' tels que $x \xrightarrow{*} z$ et $x \xrightarrow{*} z'$. On construit par récurrence une suite d'éléments $(x_n)_{n\geq 0}$ tels que $x_0 \to x_1 \to \cdots$. On pose $x_0 = x$ et on suppose avoir défini x_i tel qu'il existe deux éléments z_i et z'_i en forme normale tels que $x_i \xrightarrow{*} z_i$ et $x_i \xrightarrow{*} z'_i$. Il existe alors des éléments y et y' tels que $x_i \to y \xrightarrow{*} z_i$ et $x_i \to y' \xrightarrow{*} z'_i$. Si $y = y$', on choisit $x_{i+1} = y$. Sinon, on applique la confluence locale pour trouver z en forme normale tel que $y \xrightarrow{*} z$ et $y' \xrightarrow{*} z$. Si $z \neq z_i$, on choisit $x_{i+1} = y$, $z_{i+1} = z'_{i+1}$ et $z'_{i+1} = z$. Sinon on choisit $x_{i+1} = y'$, $z_{i+1} = z$ et $z'_{i+1} = z'_i$. □

2.7.2 Contenus de pile

Nous allons utiliser les résultats sur les relations confluentes pour montrer que les contenus de pile des configurations accessibles d'un automate à pile forment un langage rationnel. Soit un automate à pile (cf. définition 2.62) d'alphabet d'entrée A, d'alphabet de pile Z, d'état initial q_0 et de symbole de pile initial z_0. On définit le langage H de la manière suivante.

$$H = \{h \in Z^* \mid \exists f \in A^* \ \exists q \in Q \ \ (q_0, z_0) \xrightarrow{f} (q, h)\}$$

Théorème 2.81. *Pour tout automate à pile, le langage H est rationnel.*

Monoïde polycyclique

Soit A_n l'alphabet $\{a_1, \ldots, a_n, \bar{a}_1, \ldots, \bar{a}_n\}$. On définit la relation $\to$ sur A_n^* de la manière suivante. Deux mots w et w' vérifient $w \to w'$ s'il existe deux mots u et v

sur A_n et un indice i tel que $w = ua_i\bar{a}_iv$ et $w' = uv$. La relation $\to$ est noethérienne puisque $w \to w'$ implique $|w|' = |w| - 2$. On vérifie aussi facilement qu'elle est fortement confluente et donc confluente d'après la proposition 2.79. On note $\rho(w)$ l'unique mot irréductible tel que $w \xrightarrow{*} \rho(w)$. On définit alors la relation d'équivalence $\sim$ sur A_n^* par $w \sim w'$ si $\rho(w) = \rho(w')$. Cette relation est en fait une congruence sur A_n^*. Le *monoïde polycyclique* engendré par A_n est alors le monoïde quotient $A_n^*/\sim$.

Le lemme suivant établit un lien entre le monoïde polycyclique et le langage de Dyck (cf. exemple 2.8 p. 77).

Lemme 2.82. *Le langage $\{w \mid \rho(w) = \varepsilon\}$ est égal au langage de Dyck D_n^* sur n paires de parenthèses.*

Preuve. On montre par récurrence sur la longueur de la dérivation que tout mot de langage se réduit au mot vide. Inversement, on montre par récurrence sur la longueur de la réduction que tout mot qui se réduit au mot vide est dans le langage de Dyck. □

On définit la substitution σ de A_n^* dans A_n^* par $\sigma(a) = D_n^* a D_n^*$ pour toute lettre a de A_n. Le lemme suivant établit la propriété clé de cette substitution.

Lemme 2.83. *Pour tout mot $w \in A_n^*$, on a $\sigma(w) = \{w' \mid w' \xrightarrow{*} w\}$.*

Preuve. Pour tout mot $w = w_1 \cdots w_k$, on a $\sigma(w) = D_n^* w_1 D_n^* w_2 \cdots D_n^* w_k D_n^*$. Il est alors évident d'après le lemme précédent que tout mot w' de $\sigma(w)$ vérifie $w' \xrightarrow{*} w$. Inversement, on prouve par récurrence sur la longueur de la réduction de w' à w que tout mot qui vérifie $w' \xrightarrow{*} w$ est dans $\sigma(w)$. □

Corollaire 2.84. *Pour tout langage rationnel K de A_n^*, le langage $\rho(K) = \{\rho(w) \mid w \in K\}$ est rationnel.*

Preuve. Le langage $L = \{w' \mid \exists w \in K \quad w \xrightarrow{*} w'\}$ est égal à $\sigma^{-1}(K)$ et il est donc rationnel d'après la proposition 1.140. Le langage $\rho(K)$ est égal à $L \setminus \bigcup_{i=1}^{n} A_n^* a_i \bar{a}_i A_n^*$ et il est donc aussi rationnel. □

Nous sommes maintenant en mesure d'établir le théorème 2.81.

Preuve du théorème 2.81. Pour un mot $w = w_1 \cdots w_n$, on note $\tilde{w}$ le mot miroir $w_n \cdots w_1$ obtenu en renversant le mot w. On montre que le langage miroir $\tilde{H} = \{\tilde{h} \mid h \in H\}$ est rationnel, ce qui implique immédiatement que H est aussi rationnel.

Soit $\mathcal{A}$ un automate à pile. Quitte à renommer les symboles de pile, on suppose que l'alphabet de pile de $\mathcal{A}$ est $\{a_1, \ldots, a_n\}$. Soit E l'ensemble des transitions de $\mathcal{A}$. On définit un morphisme μ de E^* dans A_n^* en posant $\mu(\tau) = \bar{z}\tilde{h}$ pour toute transition τ égale à $q, z \xrightarrow{y} q', h$. Deux transitions τ et τ' de $\mathcal{A}$ sont dites consécutives si l'état d'arrivée de τ est l'état de départ de τ'. Il faut bien noter que deux transitions consécutives ne peuvent pas nécessairement être enchaînées dans un calcul. En effet, le dernier symbole de pile empilé par τ n'est pas le même que le symbole de pile dépilé par τ'. Le fait qu'elles puissent être enchaînées dépend même du contenu de pile dans le cas où le mot empilé par τ est vide. On note K l'ensemble des suites consécutives de transitions de $\mathcal{A}$. Cet ensemble est bien sûr rationnel. On a alors l'égalité

$$\tilde{H} = \rho(z_0\mu(K)) \cap \{a_1, \ldots, a_n\}^*$$

qui montre, grâce au corollaire précédent, que $\tilde{H}$ est rationnel. □

2.7.3 Groupe libre

Soit A_n l'alphabet $\{a_1, \ldots, a_n, \bar{a}_1, \ldots, \bar{a}_n\}$. La fonction $a \mapsto \bar{a}$ est étendue à tout A_n en posant $\bar{\bar{a}}_i = a_i$ pour tout $1 \leq i \leq n$. Elle devient alors une involution sur A_n.

On définit la relation $\to$ sur A_n^* de la manière suivante. Deux mots w et w' vérifient $w \to w'$ s'il existe deux mots u et v sur A_n et une lettre a de A_n tel que $w = ua\bar{a}v$ et $w' = uv$. La relation $\to$ est noethérienne puisque $w \to w'$ implique $|w|' = |w| - 2$. On vérifie aussi facilement qu'elle est fortement confluente et donc confluente d'après la proposition 2.79. L'ensemble des mots irréductibles est $I = A_n^* \setminus A_n^*(\sum_{a \in A_n} a\bar{a})A_n^*$. On note $\rho(w)$ l'unique mot irréductible tel que $w \xrightarrow{*} \rho(w)$. On définit alors la relation d'équivalence $\sim$ sur A_n^* par $w \sim w'$ si $\rho(w) = \rho(w')$. Cette relation est en fait une congruence sur A_n^*. Le *groupe libre* engendré par A_n, noté $F(A_n)$, est alors le monoïde quotient $A_n^*/\sim$. Comme chaque classe d'équivalence de $\sim$ contient un seul mot irréductible, on peut identifier les éléments du $F(A_n)$ avec l'ensemble I des mots irréductibles sur A_n. Le produit dans $F(A_n)$ est alors défini par $u, v \mapsto \rho(uv)$ pour deux mots irréductibles u et v. Pour un élément x de $F(A_n)$, c'est-à-dire une classe de $\sim$, on note $\iota(x)$ l'unique mot de x qui est irréductible. L'application ι est l'*injection canonique* de $F(A_n)$ dans A_n^*. Pour un ensemble $X \subseteq F(A_n)$, on note également $\iota(X)$ l'ensemble $\{\iota(x) \mid x \in X\}$.

La terminologie est justifiée par la propriété suivante. Pour tout mot $w = w_1 \cdots w_k$, le mot w^{-1} égal à $\bar{w}_k \cdots \bar{w}_1$ est un inverse de w dans $A_n^*/\sim$. On vérifie en effet que $\rho(ww^{-1}) = \rho(w^{-1}w) = \varepsilon$. Le monoïde $A_n^*/\sim$ est en fait un groupe.

Le corollaire 2.84 reste vrai dans le cas de la réduction qui vient d'être introduite pour définir le groupe libre. Il en découle le résultat classique suivant.

Théorème 2.85 (Benois 1969)**.** *Une partie $X \subseteq F(A_n)$ est une partie rationnelle de $F(A_n)$ si et seulement si $\iota(X)$ est une partie rationnelle de A_n^*.*

Le corollaire suivant découle immédiatement du théorème de Benois.

Corollaire 2.86. *La famille des parties rationnelles de $F(A_n)$ est close pour les opération booléennes (union, intersection et complémentation).*

Les parties reconnaissables du groupe libre sont les unions finies de classes d'un sous-groupe d'indice fini. En effet, si μ est un morphisme de monoïde de $A_n^*/\sim$ dans un monoïde fini M, alors M est nécessairement un groupe et μ est en fait un morphisme de groupe.

Sous-groupes rationnels

Lorsque le monoïde sous-jacent est un groupe G, les sous-groupes rationnels de G jouent un rôle important car ils ont la caractérisation suivante.

Proposition 2.87. *Un sous-groupe H d'un groupe G est une partie rationnelle de G si et seulement si il est finiment engendré.*

Preuve. Si le sous-groupe H est engendré par l'ensemble fini $\{g_1, \ldots, g_k\}$, alors il est égal à l'étoile K^* de l'ensemble $K = \{g_1, \ldots, g_k, g_1^{-1}, \ldots, g_k^{-1}\}$ et il est donc rationnel.

Réciproquement, soit $\mathcal{A} = (Q, G, E, I, F)$ un automate qui accepte le sous-groupe H. Un chemin dans $\mathcal{A}$ est dit *simple* s'il ne repasse pas deux fois par le même état hormis peut-être pour l'état de départ et l'état d'arrivée qui peuvent coïncider. Un chemin est

dit *presque simple* s'il se décompose $p \xrightarrow{u} q \xrightarrow{v} q \xrightarrow{w} r$ où les deux chemins $p \xrightarrow{u} q \xrightarrow{w} r$ et $q \xrightarrow{v} q$ sont simples. L'ensemble des calculs presque simples est fini puisqu'un tel chemin est de longueur au plus $2|Q|$. On note K l'ensemble fini des étiquettes des chemins réussis presque simples. On montre alors que H est engendré par K. Il est clair que $K \subseteq H$. Supposons par l'absurde que K n'engendre pas H et soit un chemin réussi c le plus court possible tel que son étiquette $h \in H$ n'est pas dans le sous-groupe engendré par K. Puisque le chemin c n'est pas simple (car sinon $h \in K$), il se décompose $p \xrightarrow{u} q \xrightarrow{v} q \xrightarrow{w} r$ où le calcul $q \xrightarrow{v} q \xrightarrow{w} r$ est presque simple. Il existe alors un calcul $i \xrightarrow{u'} q$ tel que le calcul $i \xrightarrow{u'} q \xrightarrow{v} q \xrightarrow{w} r$ soit réussi et presque simple. On a alors les égalités $h = uvw = gw(u'w)^{-1}u'vw$ qui montrent que h appartient au sous-groupe engendré par K. □

Dans le cas du groupe libre, la proposition précédente et le corollaire du théorème de Benois permettent de retrouver un résultat très classique.

Théorème 2.88 (Howson)**.** *L'intersection de deux sous-groupes finiment engendrés d'un groupe libre $F(A_n)$ est encore un sous-groupe finiment engendré.*