Mémoire de master : équations et langages formels, le cas quadratique

Michael Guedj encadré par Mr. David Renault et Mr. Géraud Sénizergues

Université Bordeaux 1

7 juin 2007

Résumé

Les équations en mots sont étudiées depuis les années 1960. Le célèbre théorème de Makanin [Mak77] établit que l'on peut décider, étant donné un système d'équations, s'il admet une solution ou non. L'objet de ce mémoire est d'étudier l'ensemble des solutions d'un système d'équations quadratique dans le monoide libre. Cet ensemble peut-être vu soit comme un ensemble de mots, que l'on appelle : ensemble des solutions-mot, soit comme un ensemble de morphismes, que l'on appelle : ensemble des solutions-morphismes. Les languages de niveau k, générés par les grammaires indexées de niveau k, ou reconnus par les automates à k-pile, sont une généralisations des languages hors-contexte. Ils forment une hierarchie stricte incluse dans les languages récursifs. Dans la première partie du mémoire, on donne les définitions de base concernant les équations en mots et les automates à k-pile. Dans la deuxième partie du mémoire, on modifie l'automate fini donné dans [Die02], de sorte à ce qu'il reconnaisse l'ensemble des solutions-morphisme d'une équation en mot quadratique. Dans la troisième partie du mémoire, on on montre que tout rationnel d'une famille d'endomorphisme de première lettre est un language de niveau 2 (ou language d'index) (résultat à rapprocher de [Cau02]). Dans la quatrième partie du mémoire, on conclut que l'ensemble des solutions-mots d'une équation en mot quadratique, est un language d'index.

Mots-clés :
Equations en mots, homomorphismes, automates, langages, combinatoire.

Table des matières

1 Introduction

Les équations en mots sont étudiées depuis les années 1960. Le célèbre théorème de Makanin [Mak77] établit que l'on peut décider, étant donné un système d'équations, s'il admet une solution ou non.

[Abd90] fournit une implémentation de l'algorithme de Makanin. [Sch92] généralise l'algorithme de Makanin (en permettant des contraintes rationnelles). [Pla04] prouve que la satisfaisabilité du problème des équations en mots est en PSPACE, et [Pla99] en NEXPTIME. Notons que la complexite exacte de l'algorithme de Makanin est inconnue. [DGH05] etablit que la theorie existentielle des equations avec contraintes rationnelles dans le groupe libre est PSPACE-complete. [RD99] montre que le probleme de satisfiabilité des equations en mots quadratiques est NP-hard. [DP02] montre, que si E est une équation en mot de taille n en une seule variable x apparaissant $\#_x$ fois dans E, alors on peut résoudre E en temps $O(n + \#_x \log n)$. [DP02] montre, en outre, que l'ensemble des solutions est soit de la forme : F, soit de la forme $F \cup (uv)^+$, où F est un ensemble de $O(\log n)$ mots, et u et v sont des mots tels que uv est un mot primitif. [DP04] montre qu'il est possible de trouver en temps $O(n^5)$ une représentation polynomiale des solutions d'une équation en mot en deux variables.

L'objet de ce mémoire est d'étudier l'ensemble des solutions d'un système d'équations quadratique (les variables apparaissent au plus deux fois) dans le monoide libre. Cet ensemble peut être vu soit comme un ensemble de mots, que l'on appelle : ensemble des solutions-mot, soit comme un ensemble de morphismes, que l'on appelle : ensemble des solutions-morphismes. La problématique s'énonce ainsi : étant donné une équation en mots quadratique, ou un système d'équations en mot quadratique, quelle est la structure, en terme de théorie des langages, de ses solutions ? Plus, précisemment, quelle est la structure de ses solutions-mot, et de ses solutions-morphisme ?

Notons que [KMP00] s'intéresse au probleme inverse, i.e. étant donné un langage, est-il l'ensemble des solutions d'une equation en mot ? Si tel est le cas, le langage est dit expressible par une equation en mot. [KMP00] montre notamment qu'il existe un hiérarchie infinie et propre de langages expressibles (basés sur le nombre de variables auxiliaires).

[Aho68] introduit les langages d'index, générés par les grammaires indexées. Ces langages s'insèrent dans la hierarchie de Chomsky, entre les langages hors-contexte et les langages récursivement énumérables. [Aho69] introduit les *nested stack automata* ; et montre que les languages d'index sont reconnu par ces automates. [Mas74] généralise les languages d'index, en introduisant les languages indexés de niveau k, obtenant ainsi une hiérarchie stricte de languages incluse dans les langages récursifs. [Mas76] généralise les automates à pile et à pile de pile, en introduisant les automates à k-pile. Les langages reconnus par les automates (non déterministes) à k-pile sont appelés : languages de niveau k. [Mas76] montre que les languages de niveau k coïncident avec les languages indéxés de niveau k. Remarquons, néanmoins, que les automates de niveau k déterministes reconnaissent les langages de niveau k déterministes. Cette notion de déterminisme est absente dans les grammaires indéxées de niveau k. Signalons que [HL] s'intéresse à ce problème de déterminisme. Ainsi les languages d'index coïncident avec les languages de niveau 2 (i.e. languages reconnus par les automates à pile de pile).

Dans la première partie du mémoire, on donne les définitions de base concernant les équations en mots et les automates à k-pile.

Dans [Die02], on donne un automate fini résolvant le problème de décidabilité de toute équation en mots quadratique. Si l'équation en mots à une solution, alors l'automate en donne une, sinon l'automate assure que l'équation n'a pas de solution. Ajoutons que les étiquettes de l'automate sont des homomorphismes. On s'attend ainsi à ce que l'ensemble des solutions-morphisme d'une équation quadratique en mots soit rationnel.

Dans la deuxième partie du mémoire, on modifie l'automate fini donné dans [Die02], de sorte à ce qu'il reconnaisse toute solution-morphisme d'une équation en mot quadratique.

On appelle endomorphisme partiellement de première lettre, les endomorphismes φ dont l'image sur toute lettre l est ϵ ou de la forme $l...$ (i.e. commençant par l). On remarque que les étiquettes de l'automate, reconnaissant toutes les solutions-morphisme d'une équation en mot quadratique, sont des endomorphisme partiellement de première lettre.

Dans la troisième partie du mémoire, on montre que tout rationnel d'une famille d'endomorphisme partiellement de première lettre, est un langage d'index (résultat à rapprocher de [Cau02]).

Dans la quatrième partie du mémoire, on conclut que l'ensemble des solutions-mots d'une équation en mot quadratique, est un langage d'index.

Première partie

Preliminaires

On donne les définitions de base concernant les équations en mots et les automates à k-pile.

2 Equation en mots

On donne les définitions de base des équations en mots, c'est-à-dire, des équations dont les constantes sont éléments d'un ensemble fini $\mathcal{A}$, appelé alphabet, et les variables, éléments d'un ensemble fini $\mathcal{V}$ (disjoint de $\mathcal{A}$), appelé ensemble des variables, définissent des éléments du monoïde libre sur $\mathcal{A} \cup \mathcal{V}$ noté $(\mathcal{A} \cup \mathcal{V})^*$. On diffère ici de [Die02] qui considère que les variables définissent des éléments du monoïde libre $\mathcal{A}^*$.

On pose : ϵ désigne le mot vide et $\leq$ l'ordre prefixe sur $(\mathcal{A} \cup \mathcal{V})^*$ $(\forall u, v \in (\mathcal{A} \cup \mathcal{V})^* : u \leq v \Leftrightarrow \exists w \in (\mathcal{A} \cup \mathcal{V})^*, u.w = v)$.

On notera les constantes par des lettres majuscules : $A, B, C, ...$ et les variables par des lettres minuscules : $x, y, z, ...$.

2.1 Equation et système d'équations en mots

Définition 2.1 (Equation en mot) *Une équation en mot, sur un alphabet $\mathcal{A}$ et un ensemble de variables $\mathcal{V}$, est un couple $(L, R) \in (\mathcal{A} \cup \mathcal{V})^* \times (\mathcal{A} \cup \mathcal{V})^*$. On note une telle équation en mot (L, R) par $L = R$.*

Définition 2.2 (Système d'équations en mot) *Un système d'équations en mot sur $(\mathcal{A} \cup \mathcal{V})^*$, est un ensemble fini, dont les élements, sont des équations en mots sur $(\mathcal{A} \cup \mathcal{V})^*$.*

Soit $E = \{(L_1, R_1), ..., (L_n, R_n)\}$ un tel système, on le note par : $(E) \begin{cases} L_1 = R_1 \\ ... \\ L_n = R_n \end{cases}$.

Dans ce mémoire, on s'interressera plus particulièrement aux équations en mots quadratiques.

2.2 Le cas quadratique

Définition 2.3 (Equation en mot quadratique) *Une équation en mot $E = (L, R)$ sur $(\mathcal{A} \cup \mathcal{V})^*$ est quadratique, si chaque variable apparaît au plus deux fois dans l'équation, i.e. si $\forall x \in \mathcal{V}, |L|_x + |R|_x \leq 2$.*

Définition 2.4 (Système d'équations en mot quadratique) *Un système d'équations en mot $E = \{(L_1, R_1), ..., (L_n, R_n)\}$, sur $(\mathcal{A} \cup \mathcal{V})^*$, est quadratique, si chaque variable apparaît au plus deux fois dans le système d'équation, i.e. si $\forall x \in \mathcal{V}$, $\sum_{i=1}^{n} |L_i|_x + |R_i|_x \leq 2$.*

2.3 Solutions-mots et solutions-morphisme

L'ensemble des solutions d'une équation en mots, ou d'un systéme d'équations en mots, peut être appréhendé comme un ensemble de morphismes ou comme un ensemble de mots.

Les solutions-morphismes sont des endomorphismes de $(\mathcal{A} \cup \mathcal{V})^*$ qui fixent l'alphabet $\mathcal{A}$ point par point, et qui font correspondre, à chaque variable, des éléments de $(\mathcal{A} \cup \mathcal{V})^*$, de manière à ce que les deux membres de l'équation (ou des équations dans le cas d'un système) soient égaux.

Si on ordonne les variables, une solution-mot est un $|\mathcal{V}|$-uplet à valeur dans $((\mathcal{A} \cup \mathcal{V})^*)^{|\mathcal{V}|}$, qui est tel que la substitution des variables par le $|\mathcal{V}|$-uplet rende l'équation (ou les équations dans le cas d'un système) vraie.

On définit ces deux approches ci-après.

Définition 2.5 (Solution-morphisme d'une équation en mot) *Soit $E = (L, R)$ une équation en mot sur $(\mathcal{A} \cup \mathcal{V})^*$. σ est une solution-morphisme de E si :*
(1) $\sigma \in End_{(\mathcal{A}\cup\mathcal{V})^*}$,
(2) σ fixe l'alphabet $\mathcal{A}$ point par point (i.e $\forall C \in \mathcal{A}, \sigma(C) = C$),
(3) $\sigma(L) = \sigma(R)$.

Définition 2.6 (Solution-morphisme d'un système d'équations en mot) *Soit $E = \{(L_1, R_1), ..., (L_n, R_n)\}$ un système d'équation en mots sur $(\mathcal{A} \cup \mathcal{V})^*$. σ est une solution-morphisme de E si :*
(1) $\sigma \in End_{(\mathcal{A}\cup\mathcal{V})^*}$,
(2) σ fixe l'alphabet $\mathcal{A}$ point par point,
(3) $\forall i \in \{1, ..., n\}, \sigma(L_i) = \sigma(R_i)$.

Si E est une équation en mots, ou un système d'équations en mots, on note $Sol_{\mathcal{E}}(E)$ l'ensemble des solutions-morphisme de E (plus précisément, ce sont des endomorphismes d'où le $\mathcal{E}$).

On pose $v = |\mathcal{V}|$ et $\mathcal{V} = \{x_1, ..., x_v\}$.

Définition 2.7 (Solution-mot d'une équation en mot) *Soit $E = (L, R)$ une équation en mot sur $(\mathcal{A}\cup\mathcal{V})^*$. $(x_1, ..., x_v) \in ((\mathcal{A}\cup\mathcal{V})^*)^v$ est une solution-mot de E si : $L(x_1, ..., x_v) = R(x_1, ..., x_v)$.*

Définition 2.8 (Solution-mot d'un système d'équations en mot) *Soit $E = \{(L_1, R_1), ..., (L_n, R_n)\}$ un système d'équations en mot sur $(\mathcal{A}\cup\mathcal{V})^*$. $(x_1, ..., x_v) \in ((\mathcal{A} \cup \mathcal{V})^*)^v$ est une solution-mot de E si : pour $i = 1, ..., n$,*
$L_i(x_1, ..., x_v) = R_i(x_1, ..., x_v)$.

Si E est une équation en mots, ou un système d'équations en mots, on note $Sol_{\mathcal{M}}(E)$ l'ensemble des solutions-mots de E.

Remarquons que $Sol_{\mathcal{E}}(E)$ et $Sol_{\mathcal{M}}(E)$ sont équipotents. En effet, l'application β, définie ci-après, est une bijection de $Sol_{\mathcal{E}}(E)$ dans $Sol_{\mathcal{M}}(E)$:

$$\beta : Sol_{\mathcal{E}}(E) \to Sol_{\mathcal{M}}(E)$$

$$\sigma \mapsto (\sigma(x_1), ..., \sigma(x_v)).$$

2.4 Exemples

Exemple 2.9 *E : $(PQ)^i x = x(QP)^i$ est une équation quadratique en mots. σ morphisme invariant sur les constantes P et Q tel que $\sigma(x) = P$ est une solution-morphisme de E. On a ici une description complète des solutions (vues comme morphismes ou comme mots) ; par exemple sous forme de mots : $Sol_{\mathcal{M}} = \{(PQ)^*P\}$.*

Exemple 2.10 *E : $x = yzA$ est une équation quadratique en mots. Le morphisme σ, invariant sur A, et vérifiant : $\sigma(x) = xyA$, $\sigma(y) = x$ et $\sigma(z) = y$ est une solution-morphisme de E. (xyA, x, y) est la solution-mot correspondante (via β). Ici, on a plus généralement $Sol_{\mathcal{M}} = \{(\alpha\beta A, \alpha, \beta) : \alpha, \beta \in \{A, x, y, z\}^*\}$.*

Exemple 2.11 $(E)\begin{cases} x_0 = A \\ y_0 = B \\ ... \\ x_i = x_{i-1}y_{i-1} \\ y_i = y_{i-1}x_{i-1} \\ ... \\ x_n = x_{n-1}y_{n-1} \\ y_n = y_{n-1}x_{n-1} \end{cases}$ *est un système d'équation en mots (non quadratique si $n > 1$). x_i correspond au i-eme terme de la suite de Thue-Morse sur $\{A, B\}$ et commençant par A, et y_i à celui commençant par B. On a : $x_1 = AB$, $x_2 = ABBA$, $x_3 = ABBABAAB$, ... et $y_1 = BA$, $y_2 = BAAB$, $y_3 = BAABABBA$, ...*

3 Automate à k-pile

On définit les automates à k-pile introduit par [Mas76]. On s'intéresse tout d'abord aux structures k-pile, puis ensuite aux automates dont la mémoire est une k-pile, appelés : automates k-pile. On utilise les définitions et notations de [Fra05] et [FS03].

3.1 Structure k-pile

Dans un premier temps, on donne la définitions des structures k-pile. On donne, ensuite, les opérations permettant de lire et modifier les structures k-piles.

Dans ce qui suit, on considère un alphabet infini :

$$\Gamma_\infty = \bigcup_{i \geq 0} \Gamma_i$$

qui vérifie :
(1) si $i \neq j$, $\Gamma_i \cap \Gamma_j = \emptyset$
(2) pour tout i, Γ_i est fini.

3.1.1 Définition

Définition 3.1 (Structure k-pile) *On définit, de manière récursive, l'ensemble des k-piles sur Γ_∞, noté pds_k (pour pushdown store), par :*
(1) $pds_0 = \{\epsilon\}$
(2) $pds_{k+1} = (\Gamma_{k+1}[pds_k])^*$.

On pose : $pds_{k\geq 0} = \bigcup_{k \geq 0} pds_k$.

Exemple 3.2 *Soit*

$$\pi = A_2[A_1[\epsilon]\ B_1[\epsilon]]\ \ B_2[B_1[\epsilon]].$$

π est une 2-pile. Pour plus de clarté on n'écrira pas les "$[\epsilon]$". La 2-pile devient :

$$\pi = A_2[A_1\ B_1]\ \ B_2[B_1].$$

D'après [Fra05] et [FS03], toute k-pile peut se représenter sous la forme d'une suite d'arbres de hauteur k. π se représente comme suit :

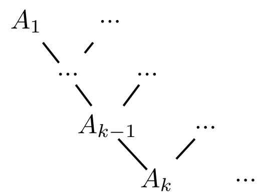
$$\pi = \begin{array}{ccc} A_1 & & B_1 \\ & \diagdown \quad \diagup & \\ & A_2 & \end{array} \quad \begin{array}{c} B_1 \\ | \\ B_2 \end{array}.$$

Remarquons que l'on utilise des symboles différents pour chaque niveau de la k-pile. Γ_i est l'alphabet des éléments empilés au niveau i de la k-pile. Dans l'exemple précédent, l'alphabet des éléments empilés au niveau 2 de la 2-pile est : $\Gamma_2 = \{A_2, B_2\}$, et celui des éléments empilés au niveau 1 est : $\Gamma_1 = \{A_1, B_1\}$.

On définit, ci-après, des applications qui vont nous permettre de lire et de modifier les k-piles. Ces applications nous permettront, dans un deuxième temps, de définir les automates à k-pile.

3.1.2 Opération sur les structures k-pile

La tête de pile d'une k-pile est la concaténation, des derniers symboles empilés, de chaque niveau de la k-pile, en suivant l'ordre décroissant des niveaux. La tête de pile d'une k-pile de la forme :



est $A_k A_{k-1} ... A_1$.

Remarquons que toute k-pile non vide : π, peut s'écrire :

$$\pi = A[\pi_1]\ \pi_2$$

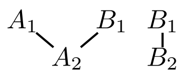
où π_1 est une $(k-1)$-pile et π_2 est une k-pile.

Définition 3.3 (Opération de lecture) *On définit l'application :*

$$top : pds_{k \geq 0} \longrightarrow \Gamma^*$$

qui associe, à chaque k-pile, sa tête de pile :
(1) $top(\epsilon) = \epsilon$
(2) $top(A[\pi_1]\ \pi_2) = A.top[\pi_1]$.

Exemple 3.4 *Soit la 2-pile :*

$\pi =$ 

On a : $top(\pi) = A_2 A_1$.

On note top_i la fonction, qui à chaque k-pile, associe le dernier symbole de pile de niveau i empilé. En reprenant l'exemple précédant, on a $top_2(\pi) = A_2$ et $top_1(\pi) = A_1$.

On définit les opérations pour modifier les k-piles. L'application identité, invariante sur toute k-pile, est notée *stay*.

L'empilement d'un symbole, au niveau i de la k-pile, se fait avec copie des piles de niveau inférieur.

Définition 3.5 (Opération d'empilement de niveau i) *Pour* $i \leq k$ *et* $S_i \in \Gamma_i$*, on définit l'opération d'empilement de niveau i :*

$$push_{i,S_i} : pds_k \longrightarrow pds_k$$

par :

(1) $push_{i,S_i}(\epsilon)$ n'est pas défini

(2) si $i = k$, $push_{i,S_i}(A_k[\pi_1]\ \pi_2) = S_i[\pi_1]\ A_k[\pi_1]\ \pi_2$

(3) si $i < k$, $push_{i,S_i}(A_k[\pi_1]\ \pi_2) = A_k[push_{i,S_i}(\pi_1)]\ \pi_2$.

Exemple 3.6 *Soit la 2-pile :*

$$\pi = \begin{array}{ccccc} A_1 & & B_1 & & B_1 \\ & \diagdown\ \diagup & & & | \\ & A_2 & & & B_2 \end{array}$$

on a

$$push_{1,S_1}(\pi) = \begin{array}{ccccccc} S_1 & & A_1 & & B_1 & & B_1 \\ & \diagdown & | & \diagup & & & | \\ & & A_2 & & & & B_2 \end{array}$$

et

$$push_{2,S_2}(\pi) = \begin{array}{ccccccccc} A_1 & & B_1 & & A_1 & & B_1 & & B_1 \\ & \diagdown\ \diagup & & & & \diagdown\ \diagup & & & | \\ & S_2 & & & & A_2 & & & B_2 \end{array}.$$

Définition 3.7 (Opération de dépilement de niveau i) *Pour $i \leq k$, on définit l'application de dépilement de niveau i :*

$$pop_i : pds_k \longrightarrow pds_k$$

par :

(1) $pop_i(\epsilon)$ n'est pas défini

(2) si $i = k$, $pop_i(A_k[\pi_1]\ \pi_2) = \pi_2$

(3) si $i < k$, $pop_i(A_k[\pi_1]\ \pi_2) = A_k[pop_i(\pi_1)]\ \pi_2$.

Exemple 3.8 *Soit la 2-pile :*

$$\pi = \begin{array}{ccccc} A_1 & & B_1 & & B_1 \\ & \diagdown\ \diagup & & & | \\ & A_2 & & & B_2 \end{array}$$

on a

$$pop_1(\pi) = \begin{array}{cc} B_1 & B_1 \\ | & | \\ A_2 & B_2 \end{array}$$

et

$$pop_2(\pi) = \begin{array}{c} B_1 \\ | \\ B_2 \end{array}.$$

Définition 3.9 (Opération d'échange de niveau i) *Pour $i \leq k$ et $S_i \in \Gamma_i$, on définit l'application d'échange de niveau i :*

$$change_{i,S_i} : pds_k \longrightarrow pds_k$$

par :

(1) $change_{i,S_i}(\epsilon)$ *n'est pas défini*

(2) *si* $i = k$, $change_{i,S_i}(A_k[\pi_1]\ \pi_2) = S_i[\pi_1]\ \pi_2$

(3) *si* $i < k$, $change_{i,S_i}(A_k[\pi_1]\ \pi_2) = A_k[change_{i,S_i}(\pi_1)]\ \pi_2$.

Exemple 3.10 *Soit la 2-pile :*

$$\pi = \begin{array}{cccc} A_1 & & B_1 & B_1 \\ \diagdown & & \diagup & \mid \\ & A_2 & & B_2 \end{array}$$

on a

$$change_{1,S_1}(\pi) = \begin{array}{cccc} S_1 & & B_1 & B_1 \\ \diagdown & & \diagup & \mid \\ & A_2 & & B_2 \end{array}$$

et

$$change_{2,S_2}(\pi) = \begin{array}{cccc} A_1 & & B_1 & B_1 \\ \diagdown & & \diagup & \mid \\ & S_2 & & B_2 \end{array}.$$

3.2 Automate à k-pile

3.2.1 Définition

Pour tout entier k, on pose :
– $TOP_k = \Gamma_k ... \Gamma_1$
– $PUSH_k = \{push_{i,S} : i \in \{0, ..., k\}, S_i \in \Gamma_i\}$
– $POP_k = \{pop_i : i \in \{0, ..., k\}\}$
– $CHANGE_k = \{change_{i,S_i} : i \in \{0, ..., k\}, S_i \in \Gamma_i\}$
– $OP_k = PUSH_k \cup POP_k \cup CHANGE_k \cup \{stay\}$.

Définition 3.11 (Automates à k-pile) *Un automates à k-pile $\mathcal{A}$, est un 7-uplet :*

$$\mathcal{A} = (\mathcal{Q}, \Sigma, \Gamma, \delta, q_0, Z, \mathcal{F})$$

où :
- $\mathcal{Q}$ *est l'ensemble fini des états*
- Σ *est l'alphabet fini d'entrée*
- Γ *est l'ensemble fini des symboles de pile*
- δ *est la fonction de transitions* $\delta : \mathcal{Q} \times (\Sigma \cup \{\epsilon\}) \times TOP_k \to \mathcal{P}(OP_k \times \mathcal{Q})$
- $q_0 \in \mathcal{Q}$ *est l'état initial de l'automate*
- $Z \in \Gamma$ *est le symbole initial de la k-pile*
- $\mathcal{F} \subseteq \mathcal{Q}$ *est l'ensemble des état finaux (i.e terminaux acceptant).*

3.2.2 Instructions composées

Les mots éléments de OP_k^* sont appelés instructions composées. Le mots $op_1.op_2...op_n$ sera vu comme l'instruction : $op_1 \circ ... \circ op_2 \circ op_n$.

Dans [Fra05], on montre que : si l'on définit les automates à k-piles, en permettant des instructions composées, satisfaisant à la contrainte suivante :

– si $op = op_1 ... op_n$, alors après chaque instruction pop_i, le reste de la suite ne contient que des instructions pop_j et $change_{j,S_j}$ avec $j \geq i$, et des instructions $push_{j,S}$ quelconques,
alors les langages reconnus par les automates, à k-piles à instructions simples, coïncident avec les langages reconnus par les automates à k-pile, à instructions composées, et satisfaisant à la contrainte ci-dessus. L'idée de la preuve, est que l'on peut simuler tout automate à instructions composées, par un automate à instructions simples. La condition posée évite la possibilité d'opérations invalides.

On choisit de travailler avec des automates à instructions composées, pour simplifier la formalisation des automates que nous définirons.

3.2.3 Sémantique

Considérons un automate à k-piles $\mathcal{A} = (\mathcal{Q}, \Sigma, \Gamma, \delta, q_0, Z, \mathcal{F})$.

On définit l'ensemble des configurations de $\mathcal{A}$ par :

$$Conf_{\mathcal{A}} = \mathcal{Q} \times \Sigma^* \times pds_k.$$

Soit

$$(q, w, \pi) \in Conf_{\mathcal{A}}$$

une configuration de $\mathcal{A}$. Alors q est l'état courant de l'automate, w est le mot restant à lire sur la bande de lecture, et π est l'état courant de la mémoire k-pile.

On définit une relation d'exécution sur $\mathcal{A}$ notée $\vdash_{\mathcal{A}}$.

Définition 3.12 (Sémantique des automates à k-piles) *La relation d'exécution de l'automate $\mathcal{A}$: $\vdash_{\mathcal{A}} \subseteq Conf_{\mathcal{A}} \times Conf_{\mathcal{A}}$ est définie par :*
$(p, w, \pi) \vdash_{\mathcal{A}} (q, w_1^{-1}w, \pi') \Leftrightarrow \exists op \in OP_k^*$*, tel que* $\left\{ \begin{array}{l} (op, q) \in \delta(p, w_1, top(\pi)) \\ \pi' = op(\pi) \end{array} \right.$,
où w_1 dénote la première lettre de w ou ϵ.

$(p, w, \pi) \vdash_{\mathcal{A}} (q, w_1^{-1}w, \pi')$ signifie que si l'on est dans la configuration (p, w, π), i.e. si l'on est dans l'état p, que le mot restant à lire sur la bande de lecture est w, et que l'état de la mémoire k-pile est π ; alors en lisant w_1, dénotant la première lettre de w ou ϵ (on parle dans ce cas d'epsilon-transition), on passe dans l'état q, et l'état de la mémoire k-pile devient π'. On définit cette relation, par l'existence d'une instruction composée transformant la k-pile π en π', et respectant les conditions relatives à la fonction de transition de l'automate.

On note $\vdash_{\mathcal{A}}^*$ la fermeture réflexive et transitive de $\vdash_{\mathcal{A}}$.

Le langage reconnu par $\mathcal{A}$ est l'ensemble des mots acceptés par l'automate. Initialement, l'état de l'automate est q_0 et l'état de la mémoire k-pile est constitué du seul symbole Z, appelé symbole de fond de pile. Le mot w est accepté par l'automate, si il existe une chaîne d'exécution qui aboutit à l'un des états finaux, en consommant entièrement la bande de lecture (c'est-à-dire en lisant le mot w) et vidant la k-pile :

$$L(\mathcal{A}) = \{w \in \Sigma^* : \exists q_f \in \mathcal{F}, (q_0, w, Z) \vdash_{\mathcal{A}}^* (q_f, \epsilon, \epsilon)\}.$$

3.3 Notation exponentielle

On choisit une autre notation pour les k-pile : la notation exponentielle. On met en exposant ce qui est en crochet. Par exemple, la 3-pile, en notation crochet,

$$A_3[A_2[C_1B_1B_1A_1]C_2[B_1]] \ \ B_3[C_2[A_1A_1A_1]] \ \ A_3[A_2[A_1]B_2[B_1]]$$

se note, en notation exponentielle,

$$A_3^{A_2^{C_1B_1B_1A_1}C_2^{B_1}} \ \ B_3^{C_2^{A_1A_1A_1}} \ \ A_3^{A_2^{A_1}B_2^{B_1}}.$$

On a donné les définitions de base des k-piles en notation crochet ; la notation exponentielle présentait, sur le plan typographique, des problèmes de confusion. En revanche, la notation exponentielle nous paraît, en particulier, bien adaptée aux 2-piles (dans ce mémoire, on ne dépassera pas le niveau 2). Il peut, toutefois, y avoir un risque de confusion dans le cas de mise en puissance. Par exemple, la 2-pile :

$$A_1^{A_2^n}$$

est ambiguë. Correspond-elle, en notation crochet, à la 2-pile :

$$(A_1[A_2])^n$$

ou bien à la 2-pile :

$$A_1[A_2^n] \quad ?$$

Ou encore, le n est-il un symbole de pile ou un entier ? On lève l'ambiguité, en posant, pour tout ensemble non vide E et tout entier n :

$$E^{(n)} = E...E$$

(E concaténé n fois), et en enconsidérant que les exposants s'appliquent aux symboles qui leurs sont le plus proches. Par exemple,

$$A_1^{A_2^{(n)}} = A_1[A_2^n]$$

et

$$(A_1^{A_2})^{(n)} = (A_1[A_2])^n.$$

En outre, la connaissance des deux notations n'est pas inutile. En effet, à partir du niveau 3, les k-piles sont difficiles à appréhender. On pourraît, par exemple, utiliser une notation qui soit un mélange des notations crochet et exponentielle. Par exemple, la 3-pile précédente pourraît se noter :

$$A_3^{A_2[C_1B_1B_1A_1]C_2[B_1]} \ \ B_3^{C_2[A_1A_1A_1]} \ \ A_3^{A_2[A_1]B_2[B_1]}$$

ou encore :

$$A_3[A_2^{C_1B_1B_1A_1}C_2^{B_1}] \ \ B_3[C_2^{A_1A_1A_1}] \ \ A_3[A_2^{A_1}B_2^{B_1}].$$

3.4 Clôture par morphisme

Les langages de niveau k (non-déterministes) forment un cône rationnel, i.e. qu'ils sont clos par homomorphisme, homomorphisme inverse et intersection avec un ensemble régulier (voir p. 124 de [Fra05]). Dans ce mémoire, on utilisera la propriété plus faible suivante.

Lemme 3.13 *Les langages de niveau k (non-déterministes) sont clos par homomorphisme.*

3.5 Exemple détaillé

$L = \{A^n\#B^n\#C^n \,:\, n \in \mathbb{N}^*\}$ est un exemple connu de langage d'index, qui ne soit pas hors-contexte. On va le prouver, en utilisant ce qui a été vu précédemment.

Pour ce faire, on va montrer qu'il existe un automate 2-pile reconnaissant L.

On distinguera 2 étapes :

(1) définition de l'automate 2-pile

(2) démonstration que l'automate défini reconnaît le language.

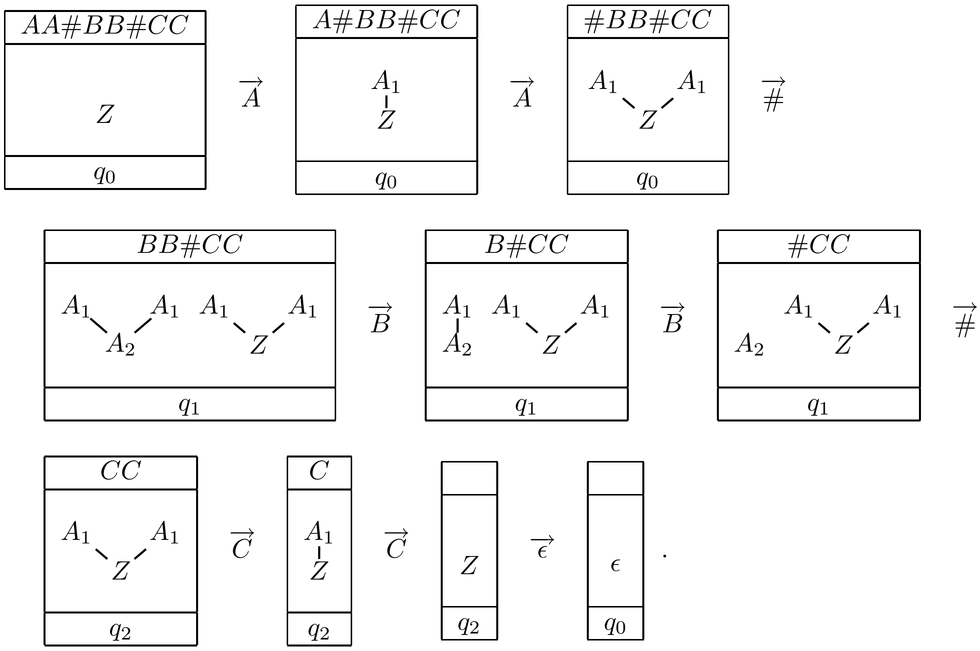
3.5.1 Définition de l'automate 2-pile

L'automate proposé, pris dans [Fra05], est basé sur l'idée suivante. A chaque lecture d'un caractère A, on empile au niveau 1 de la 2-pile un symbole A_1. A la lecture du premier symbole $\#$, on empile le symbole de niveau 2 A_2, à la lecture de chaque caractere B, on dépile un symbole de niveau 1 A_1. Une fois toutes les lettres B lues, il ne doit y avoir aucun A_1 sur le niveau 1 de la pile, si le mot passé en entrée appartient bien au langage. A la lecture du deuxième symbole $\#$, et si le niveau 1 de la 2-pile est vide, alors on dépile le symbole de niveau 2 A_2. Et de même que précèdemment, à chaque lecture d'un symbole C, on dépile au niveau 1 de la 2-pile. Si le mot passé en entrée appartient au langage, la pile de niveau 1 doit être vide. Si tel est le cas, on poursuit par une epsilon-transition qui vide la 2-pile.

On représenter l'éxécution de l'automate $\mathcal{A}$ sur le mot $A^2\#B^2\#C^2$ par un diagramme suivant, où chaque nœud est de la forme :

mot restant à lire
k-pile
état de l'automate

:



De manière équivalente, on peut représenter la chaîne d'éxécution de l'automate $\mathcal{A}$ (unique si l'automate est déterministe) en utilisant la relation d'éxécution de l'automate.

$(q_0, A^2\#B^2\#C^2, Z) \vdash_{\mathcal{A}}$

$(q_0, A\#B^2\#C^2, Z^{A_1}) \vdash_{\mathcal{A}}$

$(q_0, \#B^2\#C^2, Z^{A_1A_1}) \vdash_{\mathcal{A}}$

$(q_1, B^2\#C^2, A_2^{A_1A_1}\ Z^{A_1A_1}) \vdash_{\mathcal{A}}$

$(q_1, B\#C^2, A_2^{A_1}\ Z^{A_1A_1}) \vdash_{\mathcal{A}}$

$(q_1, \#C^2, A_2\ Z^{A_1A_1}) \vdash_{\mathcal{A}}$

$(q_2, C^2, Z^{A_1A_1}) \vdash_{\mathcal{A}}$

$(q_2, C, Z^{A_1}) \vdash_{\mathcal{A}}$

$(q_2, \epsilon, Z) \vdash_{\mathcal{A}}$

$(q_0, \epsilon, \epsilon)$.

Si le diagramme déxécution est plus parlant que la chaîne d'éxécutions, il prend, en revanche, plus de place.

On peut enfin définir l'automate $\mathcal{A}$ de manière formelle :

$$\mathcal{A} = (\{q_0, q_1, q_2\}, \{A, B, C\}, \{Z, A_1, A_2\}, \delta, q_0, Z, q_0)$$

Les états de l'automate sont q_0, q_1 et q_2, l'alphabet de la mémoire k-pile est consitué des symboles Z, A_1 et A_2 (implicitement on indice par i les symboles de niveau i), la fonction de transition est δ, l'état initial est q_0, le symbole de fond de piles est Z et l'état final est q_0. Il reste à définir la fonction de transition δ.

Lecture des A^n
(1) $\delta(q_0, A, Z) = \{(push_{1,A_1}, q_0)\}$

Lecture du premier #
(2) $\delta(q_0, \#, ZA_1) = \delta(q_0, \#, Z) = \{(push_{2,A_2}, q_1)\}$

Lecture des B^n
(3) $\delta(q_1, B, A_2A_1) = \{(pop_1, q_1)\}$

Lecture du deuxième #
(4) $\delta(q_1, \#, A_2) = \{(pop_2, q_2)\}$

Lecture des C^n
(5) $\delta(q_2, C, ZA_1) = \{(pop_1, q_2)\}$

Le mot est reconnu
(6) $\delta(q_2, \epsilon, Z) = \{(pop_2, q_0)\}$: RECONNU

Une telle définition formelle est difficile à comprendre, mais demeure nécessaire pour démontrer quel language reconnaît l'automate.

3.5.2 Démonstration que l'automate reconnaît le language

On prouve que l'automate, défini ci-avant, reconnaît effectivement le langage $L = \{A^n\#B^n\#C^n : n \in \mathbb{N}^*\}$.

Lecture des A^n
La transition (1) assure que $(q_0, A^n\#B^n\#C^n, Z) \vdash^n_{\mathcal{A}} (q_0, \#B^n\#C^n, Z^{A_1^{(n)}})$.

Lecture du premier #
La transition (2) assure que $(q_0, \#B^n\#C^n, Z^{A_1^{(n)}}) \vdash_{\mathcal{A}} (q_1, B^n\#C^n, A_2^{A_1^{(n)}} Z^{A_1^{(n)}})$.

Lecture des B^n
La transition (3) assure que $(q_1, B^n\#C^n, A_2^{A_1^{(n)}} Z^{A_1^{(n)}}) \vdash_{\mathcal{A}}^n (q_1, \#C^n, A_2 Z^{A_1^{(n)}})$.

Lecture du deuxième #
La transition (4) assure que $(q_1, \#C^n, A_2 Z^{A_1^{(n)}}) \vdash_{\mathcal{A}} (q_2, C^n, Z^{A_1^{(n)}})$.

Lecture des C^n
La transition (5) assure que $(q_2, C^n, Z^{A_1^{(n)}}) \vdash_{\mathcal{A}}^n (q_2, \epsilon, Z)$.

Le mot est reconnu
Les transitions (6) assure que $(q_2, C^n, Z) \vdash_{\mathcal{A}} (q_0, \epsilon, \epsilon)$.

On a ainsi :

$$(q_0, A^n\#B^n\#C^n, Z) \vdash_{A}^{*} (q_0, \epsilon, \epsilon).$$

ce qui ne prouve pas que l'automate défini dans 3.5.1 reconnaît effectivement le langage $L = \{A^n\#B^n\#C^n : n \in \mathbb{N}^*\}$, mais que $L \subset L(\mathcal{A})$.

On conclut par $L = L(\mathcal{A})$, en remarquant qu' à chaque fois que l'on change d'état, il nécessaire d'avoir lu un certain préfixe d'un mot élément de L ; et que pour arriver à l'état final, il est nécessaire d'avoir lu un mot élément de L.

Deuxième partie

Equation quadratique

Dans [Die02], on donne un automate fini résolvant le problème de décidabilité d'une équation en mots quadratique. Si l'équation en mots à une solution, alors l'automate en donne une, sinon l'automate assure que l'équation n'a pas de solution (arrivée dans un état terminal non acceptant). Ajoutons que l'automate est étiqueté par des morphismes. On s'attend ainsi à ce que les solutions-morphisme d'une équation en mots quadratique soit language rationnel.

Chaque état de l'automate est une équation quadratique en mots, et les transitions sont étiquetées par des morphismes. Les mots acceptés sont ainsi des éléments d'un monoïde, libre sur un ensemble fini de morphismes. Chaque mot-morphisme sera vu comme un morphisme. Plus précisemment, si $\varphi = \varphi_1 ... \varphi_n$, alors φ correspond au morphisme $\varphi_n \circ ... \circ \varphi_1$.

Dans l'automate de [Die02], tout mot accepté est une solution-morphisme de l'équation de départ.

Mais l'automate ne reconnaît pas toutes les solutions de toutes équations en mots quadratique ; par exemple toutes les solutions de $x = x$ ne sont pas reconnues.

On reprend l'automate de [Die02] en le modifiant, de sorte à ce qu'il reconnaisse l'ensemble des solutions de l'équations en mots quadratique.

Soit E une équation en mots quadratique sur $(\mathcal{A} \cup \mathcal{V})^*$. Dans un premier temps, on définit l'automate fini $\mathcal{AQ}(E)$, et on prétend qu'il reconnaît toutes les solutions-morphisme de E, c'est-à-dire que : $L(\mathcal{AQ}(E)) = Sol_{\mathcal{E}}(E)$. On démontre ensuite, en deux parties, que cet automate reconnaît effectivement toutes les solutions-morphisme de E : tout d'abord que : $L(\mathcal{AQ}(E)) \subset Sol_{\mathcal{E}}(E)$, puis ensuite que : $Sol_{\mathcal{E}}(E) \subset L(\mathcal{AQ}(E))$.

4 Automate $\mathcal{AQ}(E)$

On définit l'automate fini $\mathcal{AQ}(E)$ qui reconnaît toutes les solutions-morphisme de E, c'est-à-dire que : $L(\mathcal{AQ}(E)) = Sol_{\mathcal{E}}(E)$. On donne ensuite deux exemples.

Considérons les endomorphismes de $(\mathcal{A} \cup \mathcal{V})^*$: $x \to \epsilon$ et $x \to x\lambda$ pour $x \in \mathcal{V}$ et $\lambda \in \mathcal{A} \cup \mathcal{V}$, invariant sur $(\mathcal{A} \cup \mathcal{V}) - \{x\}$ et tels que $x \to \epsilon(x) = \epsilon$ et $x \to x\lambda(x) = x\lambda$.

Considérons maintenant l'alphabet d'entrée de l'automate, qui est un ensemble fini d'endomorphismes de $(\mathcal{A} \cup \mathcal{V})^*$:

$$\Sigma = \{Id,\ x \to \epsilon,\ x \to x\lambda\ :\ x \in \mathcal{V},\ \lambda \in \mathcal{A} \cup \mathcal{V}\}$$

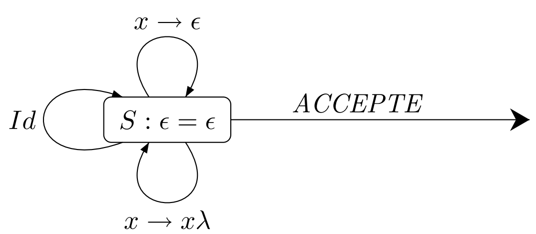
où $Id = Id_{(\mathcal{A}\cup\mathcal{V})^*}$.

Le caractère quadratique est déterminant. Etant donné une équation quadratique en mots, si on lui applique certains morphismes éléments de Σ, alors elle reste quadratique et sa taille n'augmente pas (après simplification à droite ou a gauche).

L'idée est de factoriser toute solution-morphisme d'une équation quadratique, par des éléments de Σ.

Définition 4.1 (Définition de l'automate $\mathcal{AQ}(E)$) *On définit l'automate fini :* $\mathcal{AQ}(E) = (S, \Sigma, \delta, E, \mathcal{F})$ *où :*
- *l'ensemble des états est :* $S \in (\mathcal{A}\cup\mathcal{V})^* \times (\mathcal{A}\cup\mathcal{V})^*$ *et vérifie :* $|S| \leq |L+1| \times |R+1|$, *ce qui rend l'automate fini,*
- E *est l'état initial,*
- *l'ensemble des états finaux (i.e. terminaux acceptants) est :* $\mathcal{F} = \{(\epsilon = \epsilon)\}$,
- *la fonction de transition :* $\delta : \mathcal{S} \to \mathcal{P}(\Sigma \times \mathcal{S})$ *est telle que,* $\forall S = (L, R) \in \mathcal{S}$ *:*

(1) si $S = (\epsilon = \epsilon)$, *alors* $\delta(S) = \{(x \to x\lambda,\ S), (x \to \epsilon,\ S)\ :\ x \in \mathcal{A},\ \lambda \in \mathcal{A}\cup\mathcal{V}\}$ *; on a le diagramme suivant :*



où $x \in \mathcal{V}$ *et* $\lambda \in \mathcal{A} \cup \mathcal{V}$,

(2) si $S = (\epsilon = ...A)$ *où* $A \in \mathcal{A}$, *alors* S *est un état terminal non acceptant,*

$$\boxed{S : \epsilon = ...A} \dashv$$

(3) si $S = (\epsilon = ...x)$ *où* $x \in \mathcal{V}$*, alors* $\delta(S) = \{(x \to \epsilon,\ S[x \to \epsilon])\}$,

$$\boxed{S : \epsilon = ...x} \xrightarrow{\quad x \to \epsilon \quad} \boxed{S[x \to \epsilon] : \epsilon = ...}$$

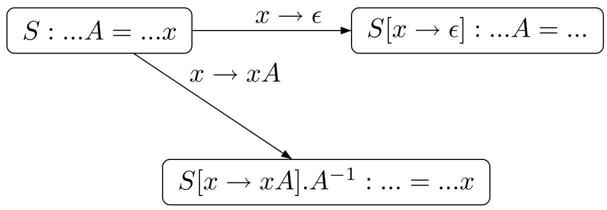
(4) si $S = (...A = ...B)$ *où* $A, B \in \mathcal{V}$*, alors*
. (a) si $A = B$*,* $\delta(S) = \{(Id,\ S.A^{-1})\}$,

$$\boxed{S : ...A = ...A} \xrightarrow{\quad Id \quad} \boxed{S.A^{-1} : ... = ...}$$

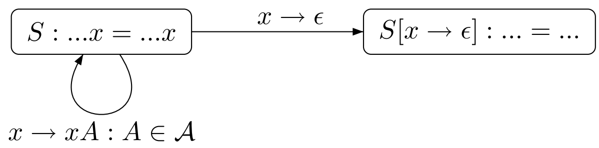
. (b) sinon, S *est un état terminal non terminant,*

$$\boxed{S : ...A = ...B} \dashv$$

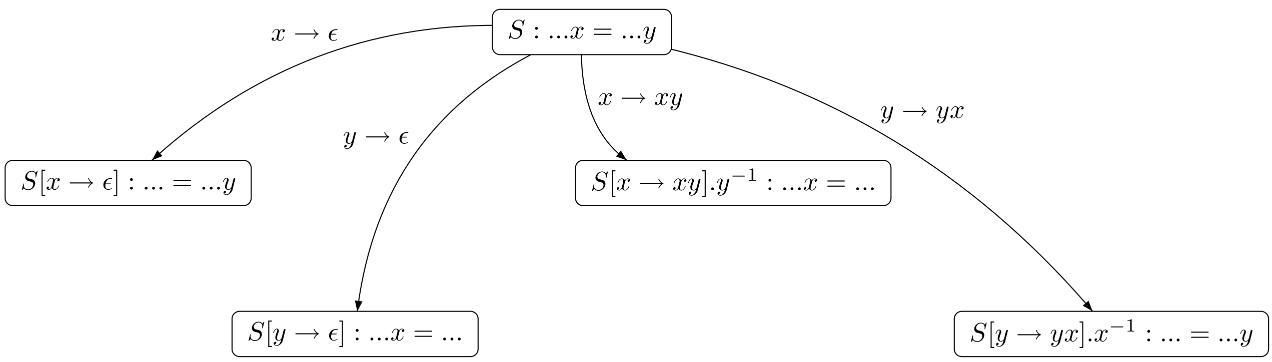
(5) si $S = (...x = ...A)$ *où* $x \in \mathcal{V}$ *et* $A \in \mathcal{A}$*, alors*
$\delta(S) = \{(x \to \epsilon,\ S[x \to \epsilon]),\ (x \to xA,\ S[x \to xA].A^{-1})\}$



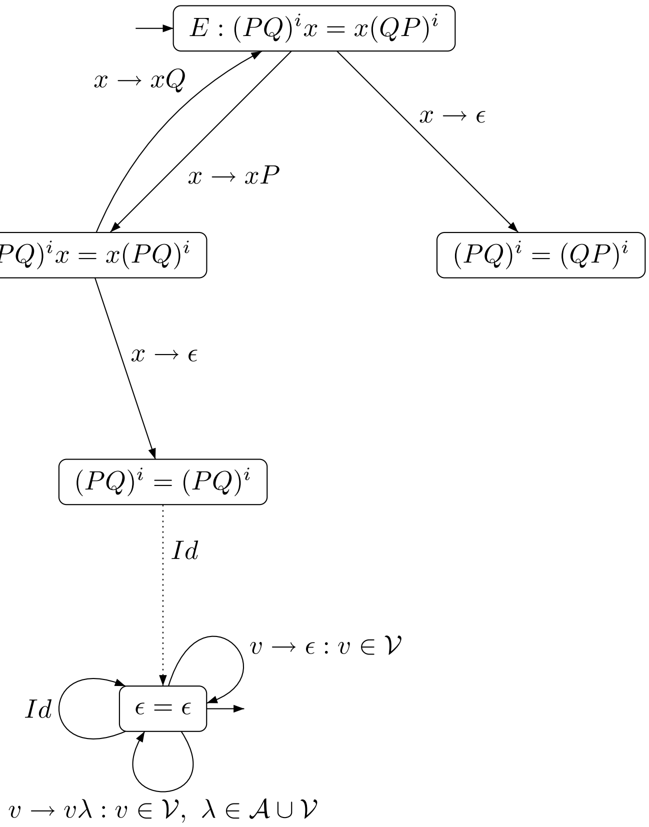
(6) si $S = (...x = ...y)$ *où* $x, y \in \mathcal{V}$, *alors*
. *(a) Si* $x = y$, $\delta(S) = \{(x \to \epsilon,\ S[x \to \epsilon]),\ (x \to x\lambda,\ S)\ :\ \lambda \in \mathcal{A} \cup \mathcal{V}\}$



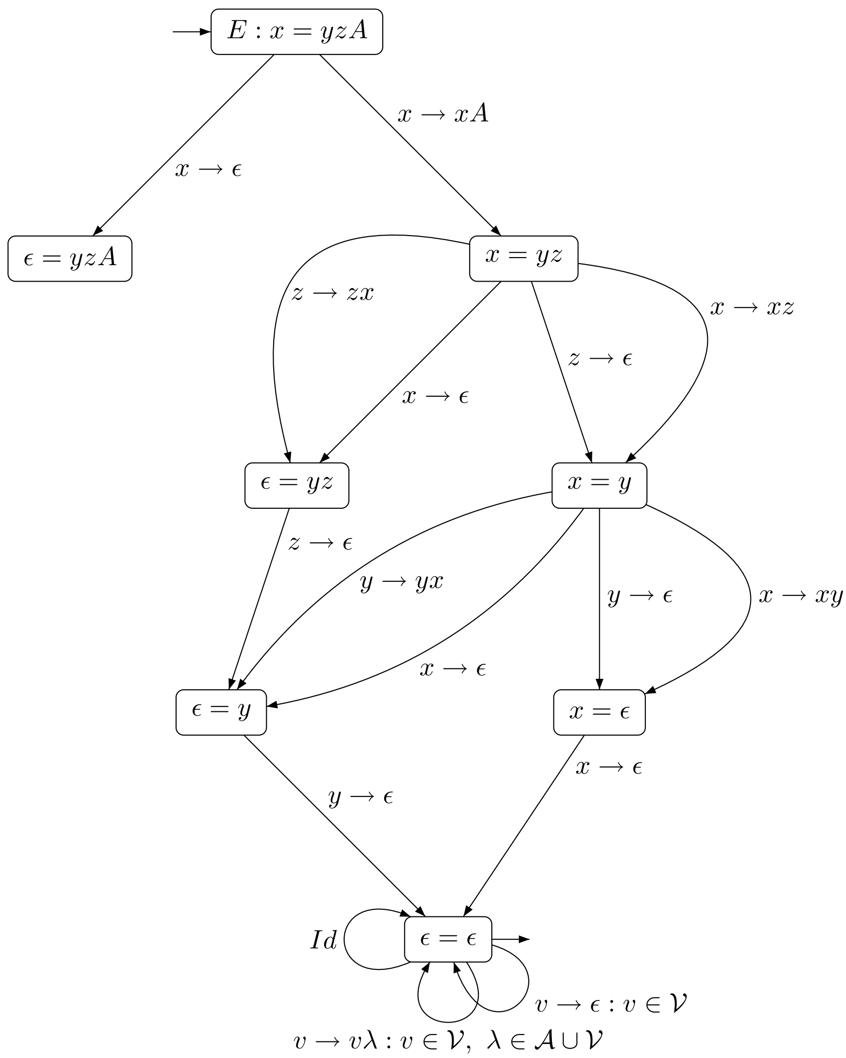
. *(b) Sinon,* $\delta(S) = \{(x \to \epsilon,\ S[x \to \epsilon]),\ (x \to xy,\ S[x \to xy].y^{-1}),\ (y \to \epsilon,\ S[y \to \epsilon]),\ (y \to yx,\ S[y \to yx].x^{-1})\}$



Exemple 4.2 $E : (PQ)^i x = x(QP)^i$ *avec* $i > 0$



Exemple 4.3 $E : x = yzA$



5 Démonstration de $L(\mathcal{AQ}(E)) = Sol_{\mathcal{E}}(E)$

Posons $\vdash$ la relation d'éxecution de l'automate $\mathcal{AQ}(E)$, définie par :

$$E_i \vdash_{h_i} E_{i+1} \Leftrightarrow (h_i, E_{i+1}) \subset \delta(E_i).$$

De manière informelle, cela signifie qu'à partir de l'état E_i, il existe une transition étiquetée par h_i, et aboutissant à l'état E_{i+1}.

Posons $L(\mathcal{AQ}(E))$ le langage accepté par l'automate $\mathcal{AQ}(E)$. Il est défini par :

$$L(\mathcal{AQ}(E)) = \{w = h_1...h_n \in \Sigma^* \ : \ \exists E_1, ..., E_{n+1} \in \mathcal{S} \mid E = E_1 \vdash_{h_1} E_2 \vdash_{h_2} ... \vdash_{h_n} E_n = (\epsilon, \epsilon)\}.$$

Le mot $w = h_1...h_n$ de Σ^* défini le morphisme $h_n \circ ... \circ h_1$; ainsi $L(\mathcal{AQ}(E))$ est bien un ensemble d' endomorphismes de $(\mathcal{A} \cup \mathcal{V})^*$.

On définit leur plus grand commun suffixe de deux mots.

Définition 5.1 (Plus grand commun suffixe de deux mots) *Soit w et w' deux mots. $pgcs(w, w')$ est le plus grand commun suffixe de w et w' si :*
(1) $pgcs(w, w')$ est un suffixe commun à w et w'
(2) si u est un suffixe commun à w et w', alors alors u est un suffixe de $pgcs(w, w')$.

Exemple 5.2 *$pgcs(ABxAB, BAxAB) = xAB$.*

On définit la relation d'équivalence $\equiv$ sur $(\mathcal{A} \cup \mathcal{V})^* \times (\mathcal{A} \cup \mathcal{V})^*$ par :

$$(L, R) \equiv (L', R') \Leftrightarrow (L, R).(pgcs(L, R))^{-1} = (L', R').(pgcs(L', R'))^{-1}.$$

Par exemple, $(ABxAB, BAxAB) \equiv (AB = BA)$.

Remarquons que $(L, R) \equiv (\epsilon, \epsilon)$ si et seulement si $L = R$, et que tout $h \in \Sigma$ preserve la relation $\equiv$ (ie : si $E \equiv E'$, alors $h(E) \equiv h(E')$).

Remarquons, en outre, que l'automate $\mathcal{AQ}(E)$ est tel que, si $E_i \vdash_{h_i} E_{i+1}$, alors $h_i(E_i) \equiv E_{i+1}$.

Proposition 5.3 *$L(\mathcal{AQ}(E)) \subset Sol_{\mathcal{E}}(E)$.*

Démonstration
Si $E = (\epsilon = \epsilon)$, tout mot morphisme reconnu est solution.
Sinon, soit $\varphi \in L(\mathcal{AQ}(E))$. Par définition, $\exists h_1, ..., h_n \in \Sigma$ et $\exists E_1, ..., E_{n+1} \in \mathcal{S}$ tel que $\varphi = h_1...h_n$ et $E = E_1 \vdash_{h_1} E_2 \vdash_{h_2} ... \vdash_{h_n} E_n = (\epsilon, \epsilon)$.

On a $\forall i \in \{1, ..., n\}, h_i(E_i) \equiv E_{i+1}$.

On prouve, par récurrence, que $\forall i \in \{1, ..., n\}, h_i \circ ... \circ h_1(E) \equiv E_{i+1}$.
– $h_1(E_1) \equiv E_2$.
– Pour $p \in \{1, ..., n-2\}$, on suppose $h_p \circ ... \circ h_1(E) \equiv E_{p+1}$.
Comme $h_{p+1}(E_{p+1}) \equiv E_{p+2}$, alors $h_{p+1} \circ ... \circ h_1(E) \equiv E_{p+2}$.

Ainsi, $\varphi(E) = h_n \circ ... \circ h_1(E) \equiv E_{n+1} = (\epsilon, \epsilon)$. Donc, si $E = (L, R)$, $\varphi(L) = \varphi(R)$ et par conséquent, φ est une solution-morphisme de E.

■

On définit un ordre sur les endomorphismes de $(\mathcal{A} \cup \mathcal{V})^*$. Soit φ et ψ deux endomorphismes de $(\mathcal{A} \cup \mathcal{V})^*$, alors $|\varphi| \leq |\psi| \Leftrightarrow \sum_{x \in \mathcal{V}} |\varphi(x)| \leq \sum_{x \in \mathcal{V}} |\psi(x)|$.

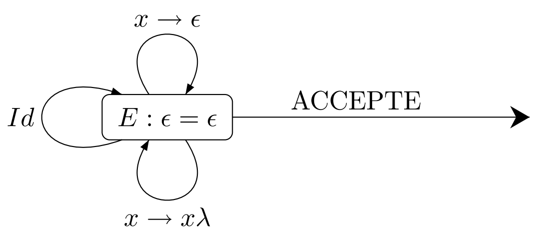
Proposition 5.4 $Sol_{\mathcal{E}}(E) \subset L(\mathcal{AQ}(E))$.

Démonstration
Soit σ une solution-morphisme de l'équation quadratique en mots : $E = (L, R)$.

(1) Si $E = (\epsilon = \epsilon)$, alors, par définition,

$$\delta(E) = \{(Id, E),\ (x \to x\lambda, E),\ (x \to \epsilon, E)\ :\ x \in \mathcal{V}\ ,\ \lambda \in \mathcal{A} \cup \mathcal{V}\}.$$

On rappelle le diagramme relatif à ce cas.



où $x \in \mathcal{V}$ et $\lambda \in \mathcal{A} \cup \mathcal{V}$.
Si $\sigma \equiv Id$, alors σ est reconnu par $\mathcal{AQ}(E)$.
Sinon, On montre que, quelle que soit la forme de l'équation en mot E, il existe un état E' de $\mathcal{S}$, un morphisme h élément de Σ, et une solution-morphisme σ' de E' qui vérifient :
(i) $\sigma = \sigma' \circ h$
(ii) $E \vdash_h E'$
(iii) $(|\sigma|, |E|) > (|\sigma'|, |E'|)$

Ainsi, par récurrence, σ est reconnu par $\mathcal{AQ}(E)$. Le cas où $E = (\epsilon, \epsilon)$ et $\sigma \equiv Id$ constitue le cas terminal de la récurrence. En effet, dans ce cas $(|\sigma|, |E|) = (0, 0)$ et σ est reconnu par l'automate.

Toujours, dans le cas (1), Si $\exists x \in \mathcal{V}, \lambda \in \mathcal{A} \cup \mathcal{V},\ \sigma(x) = ...\lambda$. On pose $\sigma' \equiv \sigma$ sur $\mathcal{V} - \{x\}$ et $\sigma'(x) = \sigma.\lambda^{-1}$, $E' = E$ et $h \equiv x \to x\lambda$.

(2) Le cas $E = (\epsilon = ...A)$ est exclus.

(3) Si $E = (\epsilon = ...x)$ où $x \in \mathcal{V}$, alors par définition, $\delta(E) = \{(x \to \epsilon, E[x \to \epsilon])\}$.

$$\begin{tikzcd} \boxed{E : \epsilon = ...x} \arrow[r, "x \rightarrow \epsilon"] & \boxed{E[x \rightarrow \epsilon] : \epsilon = ...} \end{tikzcd}$$

Nécessairement, $\sigma(x) = \epsilon$. On pose $\sigma' \equiv \sigma$, $E' = E[x \rightarrow \epsilon]$ et $h \equiv x \rightarrow \epsilon$

(4) Si $E = (...A = ...A)$ où $A \in \mathcal{V}$, alors par définition, $\delta(E) = \{(Id, E.A^{-1})\}$.

$$\begin{tikzcd} \boxed{E : ...A = ...A} \arrow[r, "Id"] & \boxed{E.A^{-1} : ... = ...} \end{tikzcd}$$

0n pose $\sigma' \equiv \sigma$, $E' = E.A^{-1}$ et $h \equiv Id$.

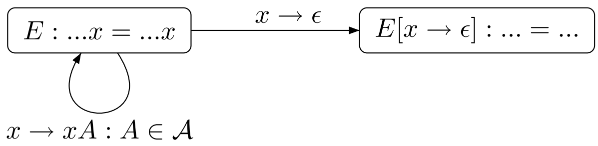
(5) Si $E = (...x = ...A)$ où $x \in \mathcal{V}$ et $A \in \mathcal{A}$, alors, par définition,
$\delta(E) = \{(x \rightarrow \epsilon, E[x \rightarrow \epsilon]),\ (x \rightarrow xA, E[x \rightarrow xA].A^{-1})\}$.

$$\begin{tikzcd} \boxed{E : ...A = ...x} \arrow[r, "x \rightarrow \epsilon"] \arrow[dr, "x \rightarrow xA"] & \boxed{E[x \rightarrow \epsilon] : ...A = ...} \\ & \boxed{E[x \rightarrow xA].A^{-1} : ... = ...x} \end{tikzcd}$$

Si $\sigma(x) = \epsilon$, alors on pose $\sigma' \equiv \sigma$, $E' = E[x \rightarrow \epsilon]$ et $h \equiv x \rightarrow \epsilon$; sinon on a $\sigma(x) = ...A$, on pose alors $\sigma' \equiv \sigma$ sur $\mathcal{V} - \{x\}$, et $\sigma'(x) = \sigma(x).A^{-1}$, $E' = E[x \rightarrow xA].A^{-1}$ et $h \equiv Id$.

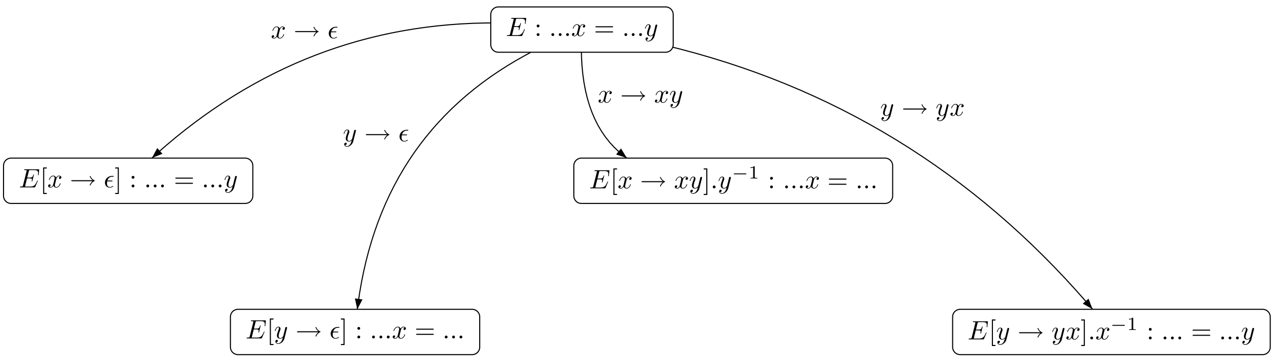
(6)
. (a) Si $x = x$ où $x \in \mathcal{V}$, alors par définition,

$$\delta(E) = \{(x \rightarrow \epsilon, E[x \rightarrow \epsilon]),\ (x \rightarrow x\lambda, E) \mid \lambda \in \mathcal{A} \cup \mathcal{V}\}.$$



Si $\sigma(x) = \epsilon$, alors on pose $\sigma' \equiv \sigma$, $E' = E[x \to \epsilon]$ et $h \equiv x \to \epsilon$; sinon $\exists \lambda \in \mathcal{A} \cup \mathcal{V} \mid \sigma(x) = ...\lambda$, alors on pose $\sigma' \equiv \sigma$ sur $\mathcal{V} - \{x\}$, et $\sigma'(x) = \sigma(x).\lambda^{-1}$, $E' = E$ et $h \equiv x \to x\lambda$.
. (b) Si $E : x = y$ où $x, y \in \mathcal{V}$ tels que $x \neq y$, alors par définition,

$\delta(E) = \{(x \to \epsilon, E[x \to \epsilon]),\ (x \to xy, E[x \to xy].y^{-1}),\ (y \to \epsilon, E[y \to \epsilon]),\ (y \to yx, E[y \to yx].x^{-1})\}$.



Si $\sigma(x) = \epsilon$, on pose $\sigma' \equiv \sigma$ et $E' = E[x \to \epsilon]$; sinon $\sigma(x) \geq \sigma(y)$, on pose $\sigma' \equiv \sigma$ sur $\mathcal{V} - \{x\}$, et $\sigma'(x) = \sigma(x).\sigma(y)^{-1}$, et $E' = E[x \to xy].y^{-1}$. (Les cas $\sigma(y) = \epsilon$ et $\sigma(y) \geq \sigma(x)$ sont symétriques respectivement, aux cas $\sigma(x) = \epsilon$ et $\sigma(x) \geq \sigma(y)$).
■

Des propositions 5.3 et 4.3, on déduit le théorème ci-après.

Théorème 5.5 *L'ensemble des solutions-morphisme, de toute équation en mots quadratique, est rationnel.*

On en déduit que l'ensemble des solutions-morphisme, de tout système d'équations en mots quadratique, est rationnel.

Corollaire 5.6 *L'ensemble des solutions-morphisme, de toute système d'équations en mots quadratique, est rationnel.*

Démonstration (Idée de démonstration)
Soit $E = \{E_1, ..., E_n\}$ un système d'équations en mots quadratique, où chaque $E_i = (L_i, R_i)$ est une équation (simple) en mots. On peut construire un automate, étiqueté sur Σ, qui reconnaît l'ensemble des solutions-morphisme de E. Considérons l'automate $\mathcal{AQ}(E_1)$, qui reconnaît l'ensemble des solutions-morphisme de E_1. A partir de $\mathcal{AQ}(E_1)$, on construit un automate fini $\mathcal{A}$ qui reconnaît l'ensemble des solutions-morphisme de E. L'état initial est $\{E_1, ..., E_n\}$. Soit un état de l'automate $\mathcal{A}$: $\{E'_1, ..., ..., E'_n\}$. Si $E'_1 \vdash_{h_1}^{\mathcal{AQ}(E_1)} E''_1$, i.e. si dans l'automate $\mathcal{AQ}(E_1)$, il existe une une transition étiquetée par h_1, qui part de E'_1 et qui aboutit à E''_1, alors nous définissons que $\{E'_1, ..., E'_n\} \vdash_{h_1}^{\mathcal{A}} \{E''_1, ..., E''_n\}$, où $E''_i = h(E'_i)$. Plus exactement, $E''_i = h(E'_i).[\ pgcs(h(L'_i), h(R'_i))\ ]^{-1}$. L'état terminal de $\mathcal{A}$ est $\{(\epsilon = \epsilon), ..., (\epsilon = \epsilon)\}$. Il reste à montrer que les équations $E''_2, ..., E''_n$ sont quadratiques et que leurs tailles est bornées.
■

On a ainsi le théorème plus général ci-après, qui est le résultat principal de la deuxième partie de ce mémoire.

Théorème 5.7 (Résultat principal de la deuxième partie du mémoire) *L'ensemble des solutions-morphisme de tout équation en mots quadratique, ou tout système d'équation en mots quadratique, est rationnel.*

Troisième partie

Composition d'endomorphismes

On appelle endomorphismes de première lettre, les endomorphismes de monoïde qui vérifient, pour toute lettre u_0 : $\varphi(u_0) = u_0 ....$

On appelle endomorphismes partiellement de première lettre, les endomorphismes de monoïde qui vérifient, pour toute lettre u_0 : $\varphi(u_0) = u_0 ...$ ou $\varphi(u_0) = \epsilon$.

Soit A un alphabet fini (et non vide). Soit X un ensemble fini (et non vide) d'endomorphismes. Supposons A et X disjoints, et considérons le symbole $\#$, qui n'appartient ni à A, ni à X.

On montre premièrement que : si X est un ensemble fini d'endomorphismes de première lettre, alors $L_1 = \{w\#w(A) : w \in X^*\}$ est un language d'index déterministe.

Ensuite, on en déduit que : si X est un ensemble fini d'endomorphismes de première lettre, et $R \in Rat(X^*)$, alors $L_2 = \{w(A) : w \in R\}$ est un language d'index.

On conclut par : si X est un ensemble fini d'endomorphismes partiellement de première lettre, et $R \in Rat(X^*)$, alors $L_3 = \{w(A) : w \in R\}$ est un language d'index.

6 Endomorphismes de première lettre

Proposition 6.1 *Si X est un ensemble fini d'endomorphismes de première lettre, alors $L_1 = \{w\#w(A) : w \in X^*\}$ est un language d'index déterministe.*

Pour prouver cette proposition, on propose un automate à 2-pile déterministe $\mathcal{A}$, qui reconnaît L'. Dans un premier temps, on définit $\mathcal{A}$. Ensuite, on démontre (de deux manières diffèrentes) que $L(\mathcal{A}) = L_1$.

6.1 Automate 2-pile $\mathcal{A}$

On propose un automate à 2-pile $\mathcal{A}$, et on prétend qu'il reconnaît $L_1 = \{w\#w(A) : w \in X^*\}$ (on le demontre plus loin).

La définition formelle de $\mathcal{A}$ étant difficile à appréhender, on donne, dans un premier temps, un diagramme représentant l'exécution (très légèrement simplifiée) de $\mathcal{A}$ sur un exemple. Ensuite, on définit formellement $\mathcal{A}$. La définition comporte la liste des transitions. On regroupe les transitions qui participe à une action informelle précisée. On appelle ces groupes les phases de l'automate. On donne un exemple détaillé de l'exécution de $\mathcal{A}$, sur le même exemple que celui utilisé pour le diagramme, en indiquant les phases et les transitions.

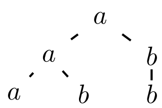
6.1.1 Exemple

On exécute l'automate $\mathcal{A}$ sur un exemple. On simplifie légèrement la suite des configurations ; on en oubliera certaines qui sont surtout d'ordre technique. On explicite ensuite les principes de fonctionnement sous-jacents à $\mathcal{A}$.

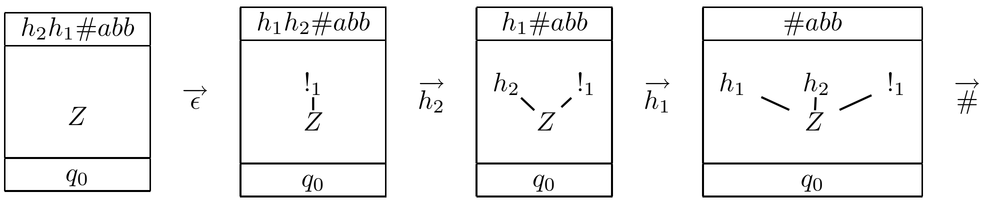
Soient les endomorphismes de $\{a, b\}^*$: h_1 et h_2 vérifiant :
- $h_1(a) = ab$,
- $h_2(a) = ab$ et $h_2(b) = b$.

h_1 et h_2 sont des endomorphismes de première lettre.

On peut représenter le mot $h_2 \circ h_1(a) = aab$ par l'abre suivant :



L'automate de niveau 2 que nous proposons s'exécutera pas à pas sur le mot $h_2h_1\#abb$ comme suit :



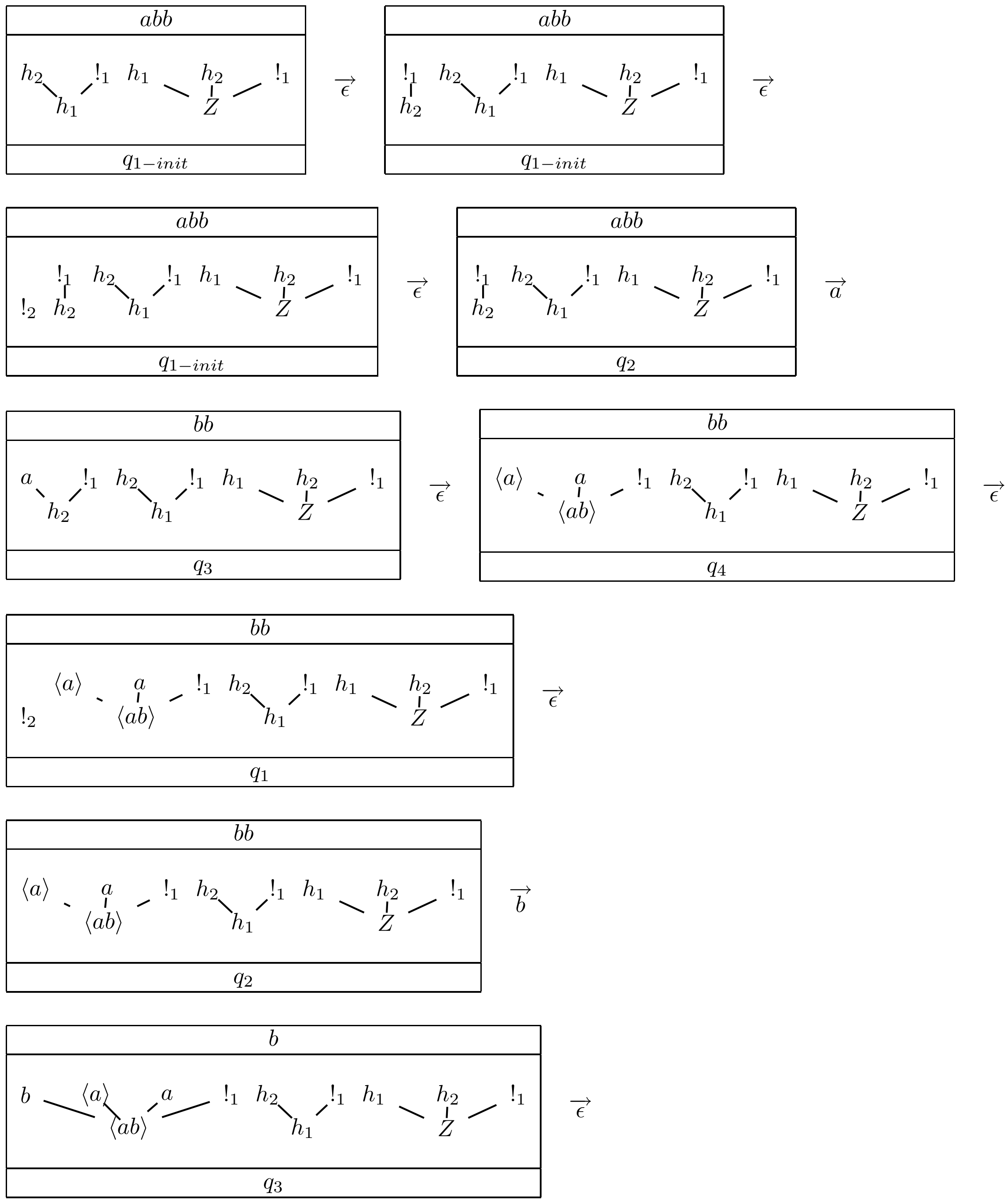
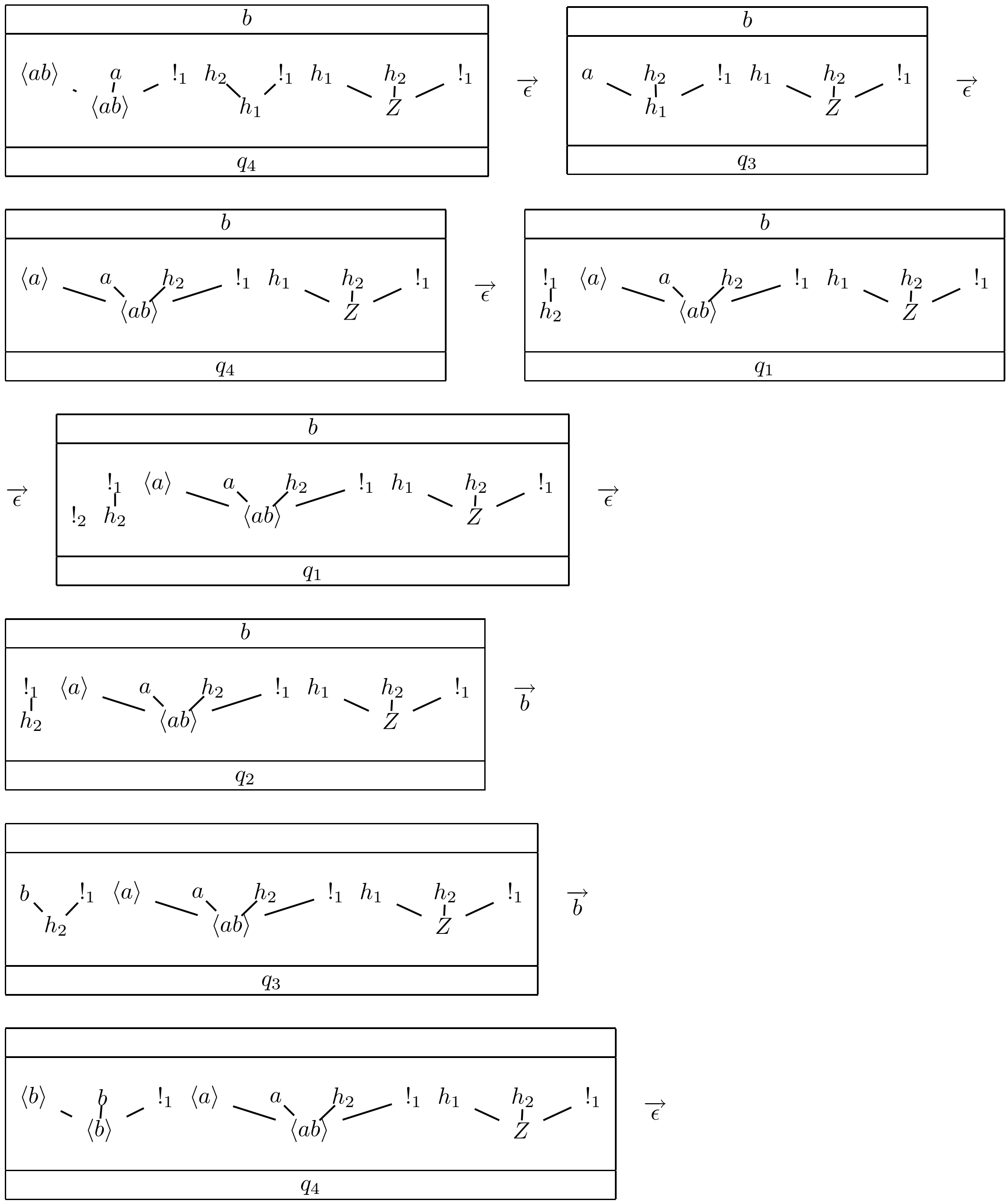
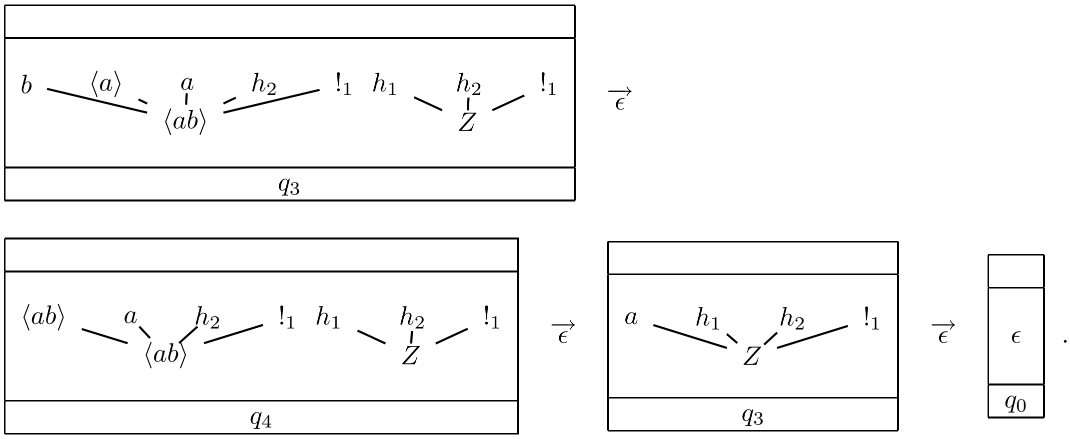


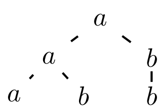
abb
q1−init
abb
q1−init
abb
q1−init
abb
q2
bb
q3
bb
q4
bb
q1
bb
q2
b
q3



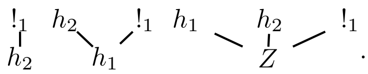
b
⟨ab⟩ a ⟨ab⟩ !1 h2 h1 !1 h1 h2 Z !1
q4
ε
b
a h2 h1 !1 h1 h2 Z !1
q3
ε
b
⟨a⟩ a h2 ⟨ab⟩ !1 h1 h2 Z !1
q4
ε
b
!1 h2 ⟨a⟩ a h2 ⟨ab⟩ !1 h1 h2 Z !1
q1
ε
b
!1 !2 h2 ⟨a⟩ a h2 ⟨ab⟩ !1 h1 h2 Z !1
q1
ε
b
!1 h2 ⟨a⟩ a h2 ⟨ab⟩ !1 h1 h2 Z !1
q2
b
b h2 !1 ⟨a⟩ a h2 ⟨ab⟩ !1 h1 h2 Z !1
q3
b
⟨b⟩ b ⟨b⟩ !1 ⟨a⟩ a h2 ⟨ab⟩ !1 h1 h2 Z !1
q4
ε



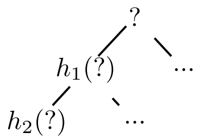
On explicite le diagramme. On rappelle que, sous forme d'arbre, le mot *aab* peut se presenter comme suit :



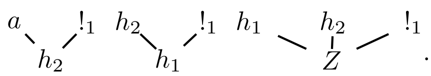
Après lecture des morphismes, l'état de la mémoire 2-pile est (modulo epsilon-transition) :



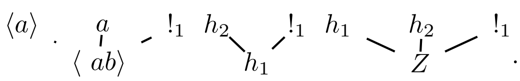
Aucunne lettre n'a encore été lue. Néamoins, nous avons des informations à priori sur le mot à lire, plus précisément sur sa structure arborescente.



Après lecture de la première lettre, la mémoire 2-pile est :.



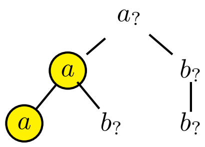
Puis après epsilon-transition :



Les chevrons $\langle\ \rangle$ sont informels, ils indiquent, que ce qui est en leur sein, doit être appréhendé comme un mot. Pour que le mot sur la bande de lecture appartienne au langage, il est nécessaire que les prochaines lettres concaténées à

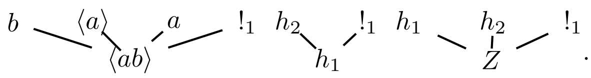
$\langle a \rangle$ soient égale à $h_2(a)$ à savoir $\langle ab \rangle$. On teste l'égalité, et si elle est fausse on redescend dans l'arborescence ; ici on est au plus bas niveau, on redescend à un niveau plancher artificiel symbolisé par le symbole $!_2$, pour remonter ensuite au niveau directement supérieur. Si l'égalité avait était vraie, on aurait remonté dans l'arborescente. Récapitulons, si l'égalité est fausse, on redescend au plus bas niveau pour remonter niveau par niveau ; et si l'égalité est vraie, on monte au niveau directement supérieur. Ici l'égalité est fausse, on descend au niveau plancher, on remonte et on continue à lire.

Construisons, au fur et à mesure, l'arbre représentant le mot en cours de lecture. A cette étape, nous avons l'information nécessaire pour connaître *a priori* quel est le mot unique qui doit être accepté par l'automate. Autrement dit et plus généralement, une fois que l'automate a lu mot-morphisme (suivi du symbole #) : $\varphi_1 ... \varphi_n\#$ et une lettre : l, alors il reconnaît seulement le mot $\varphi_n \circ ... \circ \varphi_1(l)$. L'arbre construit est :

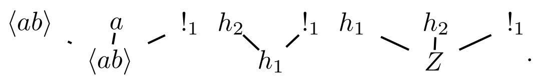


(on explicite le reste de l'arbre attendu *a priori*, en indiçant les étiquettes des sommets attendus, par des ?).

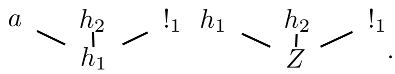
On continue la lecture sur la bande, on lit b. La mémoire 2-pile est :



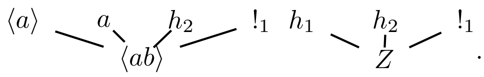
On concatène la lettre b à droite du mot $\langle a \rangle$. La mémoire 2-pile est alors :



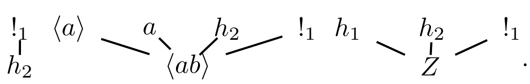
On a égalité entre le mot attendu *a priori* et le mot effectivement lu. On monte d'un niveau. La mémoire 2-pile est :



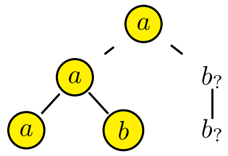
Puis après epsilon-transition :



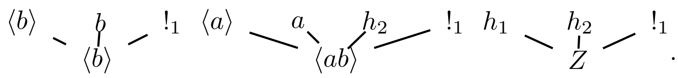
L'égalité est fausse : $\langle a \rangle \neq \langle ab \rangle$; on redescend au plus bas niveau, puis on remonte. On attend la lecture d'une lettre. La mémoire 2-pile est :



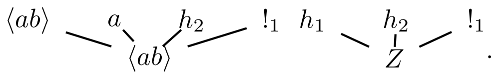
A cette étape, l'arbre construit est :



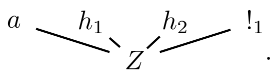
On lit la lettre suivante : b. La mémoire 2-pile est après epsilon-transition :



L'égalité est vraie, on monte d'un niveau. La mémoire 2-pile est, après epsilon-transition :

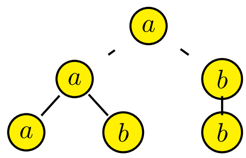


L'égalité est vraie on monte encore d'un niveau. La mémoire 2-pile est, après epsilon-transition :



Ici la tête de pile de niveau 1 est le symbole de fond de pile Z. On dépile tout, le mot est reconnu.

L'arbre représentant le mot est entierement construit :



6.1.2 Définition formelle

Si E est un ensemble, et $k \in \mathbb{N}$, on pose :
$E^{(k)}$: $E...E$ (concaténé k fois) et
$E^{\leq k} := \bigcup_{i=0,\ldots,k} E^{(i)}$.

Considérons $A = \{a_1, ..., a_k\}$, un alphabet fini et non vide ; $X = \{\phi_1, ..., \phi_{k'}\}$, un ensemble fini et non vide d'endomorphismes de A^*. Supposons, en outre, que A et X sont disjoints, et que le symbole # n'appartient ni à A, ni à X.

Pour tout mot $w = \varphi_1...\varphi_n \in X^*$, où chaque φ_i est dans X, on prétend que l'automate $\mathcal{A}$, définit ci après,reconnaît l'image de la composition de morphismes $\varphi_n \circ ... \circ \varphi_1$ sur A.

Définition 6.2 (Définition formelle de l'automate $\mathcal{A}$)
$\mathcal{A} = (\mathcal{Q}, \Sigma, \Gamma, \delta, q_0, Z, q_0)$ *où :*

- *l'ensemble des état est :* $\mathcal{Q} = \{q_0, q_1, q_{1-init}, q_{1-temp}, q_2, q_3, q_4, q_{4-temp}\}$

- *l'alphabet d'entrée est :* $\Sigma = \{a_1, ..., a_k, \#, \phi_1, ..., \phi_{k'}\}$,

- *l'alphabet de 2-pile est :*

$\Gamma = \{\phi_1^2, ..., \phi_{k'}^2, \phi_1^1, ..., \phi_{k'}^1, \} \cup \{a_1^2, ..., a_k^2\}^{\leq MAX} \cup \{a_1^1, ..., a_k^1\}^{\leq MAX} \cup \{!_1, !_2, \$_2\}$

où :
– $MAX = max\{|\varphi(l)| : \varphi \in V, l \in A\}$.
– a_i^2, *repsectivement* a_i^1, *dénote le symbole de pile de niveau 2 représentant la lettre* a_i, *respectivement le symbole de pile de niveau 1 représentant la lettre* a_i.
– ϕ_i^2, *repsectivement* ϕ_i^1, *dénote le symbole de pile de niveau 2 représentant le morphisme* ϕ_i, *respectivement le symbole de pile de niveau 1 représentant la lettre* ϕ_i.
– $!_1$ *est un symbole spécial de niveau 1, et* $!_2$ *et* $\$_2$ *sont des symbomles spécials de niveau 2. Informellement, ! est un plancher et \$ est un symbole temporaire.*

Considérons φ *et* ψ *deux morphismes éléments de* V, φ_1, ψ_1, φ_2 *et* ψ_2 *les symboles de 1-pile et 2-pile correspondants,* C *une lettre de l'alphabet d'entrée,* C_1 *et* C_2 *les symboles de 1-pile et 2-pile correspondants,* t_1 *et* t'_1 *des symboles de 1-pile,* t_2 *et* t'_2 *des symboles de 1-pile.*

On se donne une fonction de normalisation : n. *Les symboles* t_i, *respectivement* t'_i, *correspondent au symbole normalisé* t, *respectivement* t'. *On a* $n(t_i) = t$ *et* $n(t'_i) = t'$. *Par exemple,* $n(C_2) = n(C_1) = C$.

On rappelle qu'un mot constitué de la concaténation d'operateurs de pile : $op = op_1...op_k \in OP^*$, *correspond à l'operateur composé :* $op = op_k \circ ... \circ op_1$.

- *La fonction de transition* δ *est telle que :*

A - Mise en place du plancher
$(A.1)$ $\delta(q_0, \epsilon, Z) = \{(push_{1,!_1}, q_0)\}$

B - Lecture des morphismes
$(B.1)$ $\delta(q_0, \varphi, Z!_1) = \delta(q_0, \varphi, Z\psi_1) = \{(push_{1,\varphi_1}, q_0)\}$

C - Lecture du symbole #
$(C.1)$ $\delta(q_0, \#, Z!_1) = \{(pop_1.push_{2,!_2}, q_{1-init})\}$

$(C.2)$ $\delta(q_0, \#, Z\psi_1) = \{(pop_1.push_{2,\psi_2}, q_{1-init})\}$

D - Construction de la configuration initiale
$(D.1)$ $\delta(q_{1-init}, \epsilon, t_2t'_1) = \{(push_{2,\$_2}.change_{2,t'_2}.pop_1, q_{1-init})\}$

E - Remontée au niveau le plus bas
$(E.1)$ $\delta(q_{1-init}, \epsilon, !_2) = \{(pop_2, q_2)\}$

F - Descente jusqu'au plancher
$(F.1)$ $\delta(q_1, \epsilon, t_2t'_1) = \{(push_{2,\$_2}.pop_1.pop_1, q_{1-tmp})\}$

$(F.2)$ $\delta(q_{1-tmp}, \epsilon, t_2 t'_1) = \{(change_{2,t'_2}.pop_1, q_1)\}$
(Rappel : $n(t'_1) = n(t'_2) = t'$.)

G - Remontée au niveau le plus bas
$(G.1)$ $\delta(q_1, \epsilon, !_2) = \{(pop_2, q_2)\}$

H - Lecture d'un caractère sur la bande de lecture
$(H.1)$ $\delta(q_2, C, t_2 t'_1) = \{(push_{1,C_1}, q_3)\}$

I - Modification de la configuration. Au terme de cette étape, on arrive ensuite à un check-point.
$(I.1)$ $\delta(q_3, \epsilon, \psi_2 C_1) = \{(push_{1,\langle C_1 \rangle}.change_{2,\langle \psi_2(C_2) \rangle}, q_4)\}$

$(I.2)$ $\delta(q_3, \epsilon, t_2 C_1) = \{(change_{1,\langle top_1 \bullet C_1 \rangle}.pop_1, q_4)\}$
top_1 signifie tête de pile de niveau 1 courante, i.e. ici la tête de pile de niveau 1, après le pop_1. Cette opération n'est pas légale au sens strict ; néanmoins, on peut la simuler via l'ajout d'un nouvel état (dépendant du caractère C).

On explicite cette transition. On dépile la tête de pile de niveau 1 (pop_1) et on modifie la nouvelle tête de pile de niveau 1, soit top_1, par top_1 concaténée avec C_1 : $\langle top_1 \bullet C_1 \rangle$. Les chevrons sont informels, ils aident simplement à la lecture. Ils indiquent que le symbole qui est en leurs seins (ici $top_1 \bullet C_1$) est un mot (ou peut être un mot).

J - Arrivée à un check-point : test d'égalité
$(J.1)$ $\delta(q_4, \epsilon, t_2 t'_1) = \{(stay, q_1)\}$ *où* $t \neq t'$.
Le test d'égalité est faux. La tête de pile de niveau 1 normalisée n'est pas égale à la tête de pile de niveau 2 normalisée ($n(t'_1) = t' \neq t = n(t_2)$). On va à l'étape F (descente jusqu'au plancher).

$(J.2)$ $\delta(q_4, \epsilon, t_2 t_1) = \{(pop_1, q_{4-tmp})\}$
Le test d'égalité est vrai. La tête de pile de niveau 1 normalisée est égale à la tête de pile de niveau 2 normalisée ($n(t_1) = n(t_2) = t$).

$(J.3)$ $\delta(q_{4-tmp}, \epsilon, t_2 t'_1) = \{(push_{1,t'_1}.pop_2, q_3)\}$
On va à l'étape I (modification de la configuration) au terme de laquelle, on arrive de nouveau à un check-point.

K - Mot reconnu
$(K.1)$ $\delta(q_3, \epsilon, Z t_1) = \{(pop_2, q_0)\}$.

On donne un exemple détaillé de l'exécution de $\mathcal{A}$, sur le même exemple que celui utilisé pour le diagramme ($h_2 h_1 \# abb$), en indiquant les phases et les transitions.

Exemple 6.3

A - Mise en place du plancher
$(q_0,\ h_2h_1\#abb,\ Z) \vdash_A$ *(A.1)*

B - Lecture des morphismes
$(q_0,\ h_2h_1\#abb,\ Z^{!_1}) \vdash_A$ *(B.1)*

$(q_0,\ h_1\#abb,\ Z^{h_2!_1}) \vdash_A$ *(B.1)*

C - Lecture du symbole #
$(q_0,\ \#abb,\ Z^{h_1h_2!_1}) \vdash_A$ *(C.1)*

D - Construction de la configuration initiale
$(q_{1-init},\ abb,\ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(D.1)*

$(q_{1-init},\ abb,\ h_1^{h_2!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(D.1)*

$(q_{1-init},\ abb,\ h_2^{!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(D.1)*

E - Remontée au niveau le plus bas
$(q_{1-init},\ abb,\ !_2\ \ h_2^{!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(E.1)*

H - Lecture d'un caractère sur la bande de lecture
$(q_2,\ abb,\ h_2^{!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(H.1)*

I - Modification de la configuration. Au terme de cette étape, on arrive ensuite à un check-point.
$(q_3,\ bb,\ h_2^{a_1!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(I.1)*

J - Arrivée à un check-point : test d'égalité
$(q_4,\ bb,\ \langle a_1b_1\rangle^{a_1!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(J.1)*

F - Descente jusqu'au plancher
$(q_1,\ bb,\ \langle a_1b_1\rangle^{\langle a_1\rangle a_1!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(F.1)*

$(q_{1-tmp},\ bb,\ \$_2^{!_1}\ \ \langle a_1b_1\rangle^{\langle a_1\rangle a_1!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(F.2)*

G - Remontée au niveau le plus bas
$(q_1,\ bb,\ !_2\ \ \langle a_1b_1\rangle^{\langle a_1\rangle a_1!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(G.1)*

H - Lecture d'un caractère sur la bande de lecture
$(q_2,\ bb,\ \langle a_1b_1\rangle^{\langle a_1\rangle a_1!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(H.1)*

I - Modification de la configuration. Au terme de cette étape, on arrive ensuite à un check-point.

$(q_3,\ b,\ \langle a_1b_1\rangle^{b_1\langle a_1\rangle a_1!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(I.2)*

J - Arrivée à un check-point : test d'égalité
$(q_4,\ b,\ \langle a_1b_1\rangle^{\langle a_1b_1\rangle a_1!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(J.2)*

$(q_{4-tmp},\ b,\ \langle a_1b_1\rangle^{a_1!_1}\ \ h_1^{h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(J.3)*

I - Modification de la configuration. Au terme de cette étape, on arrive ensuite à un check-point.
$(q_3,\ b,\ h_1^{a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(I.1)*

J - Arrivée à un check-point : test d'égalité
$(q_4,\ b,\ \langle a_1b_1\rangle^{\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(J.1)*

F - Descente jusqu'au plancher
$(q_1,\ b,\ \langle a_1b_1\rangle^{\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(F.1)*

$(q_{1-tmp},\ b,\ \$_2^{h_2!_1}\ \ \langle a_1b_1\rangle^{\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(F.2)*

$(q_1,\ b,\ h_2^{!_1}\ \ \langle a_1b_1\rangle^{\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(F.1)*

$(q_1,\ b,\ \$_2^{!_1}\ \ h_2^{!_1}\ \ \langle a_1b_1\rangle^{\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(F.2)*

G - Remontée au niveau le plus bas
$(q_1,\ b,\ !_2\ \ h_2^{!_1}\ \ \langle a_1b_1\rangle^{\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(G.1)*

H - Lecture d'un caractère sur la bande de lecture
$(q_2,\ b,\ h_2^{!_1}\ \ \langle a_1b_1\rangle^{\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(H.1)*

I - Modification de la configuration. Au terme de cette étape, on arrive ensuite à un check-point.
$(q_3,\ \epsilon,\ h_2^{b_1!_1}\ \ \langle a_1b_1\rangle^{\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(I.1)*

J - Arrivée à un check-point : test d'égalité
$(q_4,\ \epsilon,\ \langle b_1\rangle^{\langle b_1\rangle b_1!_1}\ \ \langle a_1b_1\rangle^{\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(J.2)*

$(q_{4-tmp},\ \epsilon,\ \langle b_1\rangle^{b_1!_1}\ \ \langle a_1b_1\rangle^{\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(J.3)*

I - Modification de la configuration. Au terme de cette étape, on arrive ensuite à un check-point.
$(q_3,\ \epsilon,\ \langle a_1b_1\rangle^{b_1\langle a_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(I.2)*

J - Arrivée à un check-point : test d'égalité
$(q_4,\ \epsilon,\ \langle a_1b_1\rangle^{\langle a_1b_1\rangle a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(J.2)*

$(q_{4-tmp},\ \epsilon,\ \langle a_1b_1\rangle^{a_1h_2!_1}\ \ Z^{h_1h_2!_1}) \vdash_A$ *(J.3)*

K - Mot reconnu
$(q_3,\ \epsilon,\ Z^{a_1 h_1 h_2 !_1}) \vdash_A$ *(K.1)*

$(q_0,\ \epsilon,\ \epsilon)$.

7 Première démonstration

7.1 Configuration simplifiée de la memoire 2-pile aux configurations stables

La mémoire 2-pile est une suite d'arbres de hauteur 1 (mis à part l'arbre-plancher représenter par $!_2$). Lorsque l'automate a fait le plus d'epsilon-transition possible, ou qu'il a consommé entièrement la bande de lecture, on dit qu'il est en configuration stable. Notons que dans toute configuration stable, l'arbre-plancher ($!_2$) n'est pas présent.

L'automate $\mathcal{A}$ est en position stable, lorsqu'il attend la lecture d'une lettre, ou lorsque le mot passé sur la bande de lecture est consommé.

Tous les arbres constituant la mémoire 2-pile, sauf le premier (contenant le symbole de fond de pile Z), sont de la même forme aux positions stables.

Plus précisément, supposons que l'automate reconnaisse l'image de la composition de morphismes : $\varphi_n \circ ... \circ \varphi_1$. Alors chaque arbre considéré est du type :

$$\begin{array}{cccccc} \langle CDE...\rangle & C & \varphi_{i+1} & \cdots & \varphi_n & !_1 \\ & & \varphi_i(C) & & & \end{array}.$$

Le mot attendu est $\varphi_i(C)$, on a lu $\langle CDE...\rangle$ (pas d'une seule traite si $i \neq n$). On sauvegarde également la lettre C sur laquelle s'applique le morphisme φ_i, les morphismes de niveau "supérieur" : $\varphi_{i+1}, ..., \varphi_n$, ainsi que le plancher $!_1$. La sauvegarde de la lettre sur laquelle s'applique le morphisme (ici C) n'est pas nécessaire, mais plutôt d'ordre pratique. En effet, le morphisme étant de première lettre, on sait à priori que la lettre sur laquelle s'applique le morphisme est le premier caractère du mot $\langle CDE...\rangle$. Ainsi, on oubliera cette sauvegarde pour l'étude de l'automate. Les morphismes sauvegardés $\varphi_{i+1}, ..., \varphi_n$, ainsi que le plancher $!_1$, sont nécessaires au bon fonctionnement de l'automate. Cependant, pour l'étude de l'automate, nous les oublions, car si on ordonne les morphismes, ils n'apportent aucune information. A la place, de $\varphi_i(C)$, nous marquerons simplement φ_i.

En outre, on simplifie l'arbre $\begin{array}{cccc} \varphi_1 & \cdots & \varphi_n & !_1 \\ & Z & & \end{array}$ par Z.

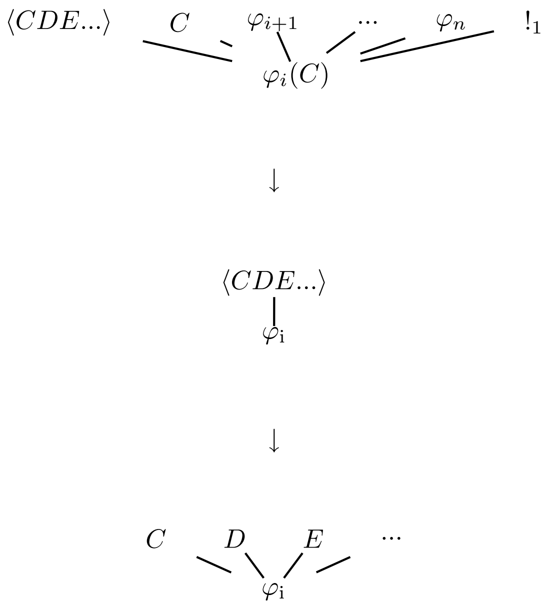
Ainsi, il existe une bijection entre les suites d'arbres représentant la mémoire 2-pile aux configurations stables, et des suites d'arbres simplifiés, où chaque arbre est du type :

$$\begin{array}{c} \langle CDE...\rangle \\ | \\ \varphi_\text{i}(\text{C}) \end{array}.$$

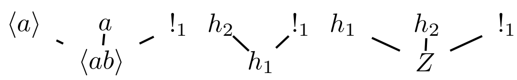
Si on ordonne les lettres du mot lu, on peut considéré des suites d'arbres où chaque arbre, sauf le premier, est du type :

$$\begin{array}{cccc} C & D & E & \cdots \\ & \varphi_\text{i}(\text{C}) & & \end{array}.$$

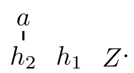
Récapitulons, nous avons défini (informellement) des bijections entre les configurations stables de la mémoire 2-pile et des configurations simplifiée. On résume les transformations subies par les arbres constituant la mémoire 2-pile :



Exemple 7.1 *La configuration :*



est simplifiée en :

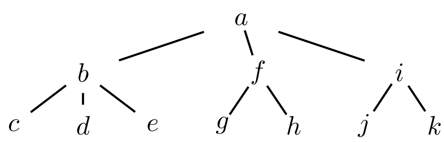


7.2 Préliminaires combinatoires à la première démonstration

Soit T un arbre enraciné. On pose : V_T l'ensemble des sommets de T, $E_T \subset V_T \times V_T$ l'ensemble des arêtes de T. $RACINE_T$ la racine de T et $FRANGE_T$ la concaténation des feuilles de l'arbre T suivant l'ordre préfixe. On pose, pour tout sommet $s \in V_T$, $NIVEAU(s)$ le niveau du s, c'est-à-dire la longeur du plus court chemin entre s et $RACINE_T$; $PERE(s)$ le père de s (on pose $PERE(RACINE_T) = Z$).

Exemple 7.2 *Si*

$T =$



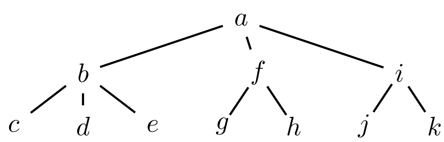
.

On a $RACINE_T = a$, $FRANGE_T = cdeghjk$, $NIVEAU(a) = 0$, $NIVEAU(b) = 1$, $NIVEAU(c) = 2$ *et* $PERE(i) = a$.

Définition 7.3 (Morphisme des fils) *Soit $FILS$ l'endomorphisme de $V(T)^*$, qui pour tout sommet s, est invariant si s est une feuille ; et sinon qui associe la concaténation, suivant l'ordre préfixe dans l'arbre T, de ses fils. Si l'arbre T est de hauteur n, alors $FILS^n(RACINE_T) = FRANGE_T$*

Exemple 7.4 *Si*

$T =$



.

On a $FILS(a) = bfi$ *et* $FILS^2(a) = cdeghjk = FRANGE_T$.

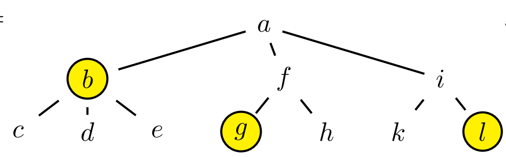
On dit que T est parfaitement équilibré si toutes les feuilles de T sont au même niveau. L'arbre de l'exemple précédent est parfaitement équilibré. Dans ce qui suit, on considère un arbre T parfaitement équilibré.

On pose $\prec_T$ l'ordre préfixe sur les sommets de T.

Définition 7.5 (Transversale gauche partielle : tgp) *Le mot $u = u_1...u_k$ est une transversale gauche partielle de T si :*
(1) $u_1 \prec_T ... \prec_T u_k$
(2) $i < j \Rightarrow level(u_i) \leq level(u_j)$
(3)$\forall i, PERE(u_i) \notin \{u_1, ..., u_k\}$.

Définition 7.6

$T =$



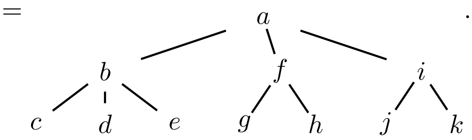
.

bgl est une transversale gauche partielle de T.

Définition 7.7 (Morphisme des feuilles tombantes) *Soit FT l'endomorphisme de V_T^*, dit des feuilles tombantes, défini par : pour tout sommets s,*
(1) si s est une feuille, $FT(s) = s$
(2) sinon, $FT(s) = FT(FILS(s))$.

Remarque 7.8 *On a* $FT(RACINE_T) = FRANGE_T$.

Exemple 7.9 *Si*

$T =$  .

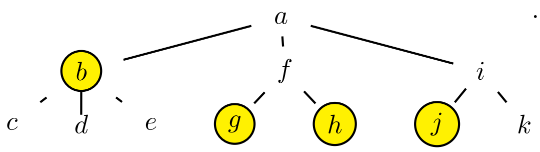
On a :

$$FT(a) = FT(b).FT(f).FT(i) = cde.FT(f).FT(i) = cde.gh.FT(i) = cdeghjk.$$

On dit qu'un mot est un pré-frange de T, si il est préfixe de $FRANGE_T$. Sur l'arbre de l'exemple précédent, *cdeg* est un pré-frange.

Définition 7.10 (Transversale gauche : tg) *Le mot* u *est une transversale gauche de* T *si :*
(1) u *est une transversale gauche partielle de* T
(2) $FT(u)$ *est un pré-frange de* T.

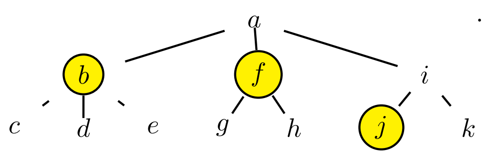
Exemple 7.11

$T =$  .

bghj est une transversale gauche de T. $FT(bghj) = cdeghj$ *est bien un pré-frange de* T.

Définition 7.12 (Transversale gauche minimale : tgm) *Le mot* $u = u_1...u_k$ *est une transversale gauche minimale de* T *si :*
(1) u *est une transversale gauche de* T
(2) $\forall i, j \in \{1, ..., k\}, i \leq j, \forall s \in V_T, FILS(s) \neq u_i...u_j$. *Autrement dit, tout facteur de* u *n'est pas égal à la concaténation (suivant* $\prec_T$*) des fils d'un sommet de* T.

Exemple 7.13

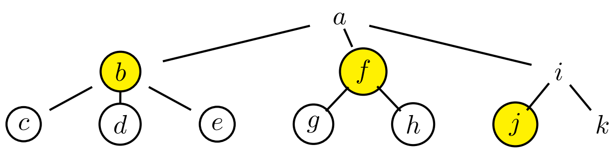
$T =$  .

bfj est une transversale gauche minimale de T.

Définition 7.14 (tg associée à un pré-frange) *t est une transversale gauche associée à un pré-frange p, si*
(1) t est une transversale gauche
(2) $FT(t) = p$.

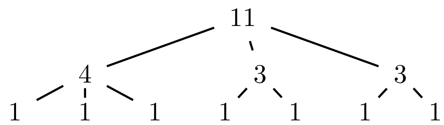
On pose tgm_p la transversale gauche minimale associée à p, et TG_p l'ensemble des transversales gauches associées à p.

Exemple 7.15

$T =$  .

$cdeghj$ est un pré-frange de T. On a : $TG_{cdeghj} = \{cdeghj, bghj, bfj\}$ et $tgm_{cdeghj} = bfj$.

On considère la fonction de poids $P : V_T \to \mathbb{N}$ définie par :
(1) si s est une feuille, $P(s) = 1$
(2) sinon, $P(s) = 1 + \sum_i P(FILS(s)[i])$.
On marque les poids de chaque sommet de l'arbre de l'exemple précèdant :

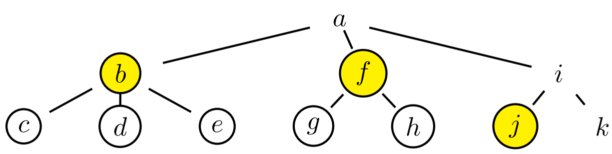
$T =$  .

Définition 7.16 (Ordre sur les tg) *On définit l'ordre $\sqsubset$ sur les transversales gauches. Soit $\alpha = \alpha_1 ... \alpha_k$ et $\beta = \beta_1 ... \beta_{k'}$ deux transversales gauches de T, alors :*

$$\alpha \sqsubset \beta \Leftrightarrow \sum_{i=1}^{k} P(\alpha_i) > \sum_{i=1}^{k'} P(\beta_i).$$

Remarque 7.17 *(1) $\sqsubset$ n'est pas un odre total sur TG_p, cependant $(TG_p, \sqsubset)$ possède un unique minimum : tgm_p.*
(2) Si on pose TGM_T l'ensemble des tgm de T, alors $\sqsubset$ est totale sur TGM_T. On remarque, en outre, que si $t_1, t_2 \in TGM_T$ alors : $t_1 \sqsubset t_2 \Leftrightarrow FT(t_1) > FT(t_2)$.

Exemple 7.18

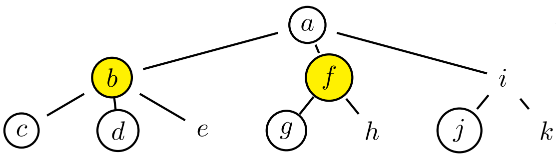
$T =$  .

$TG_{cdeghj} = \{cdeghj, bghj, bfj\}$ *et* $tgm_{cdeghj} = bfj$. *On a :*

$$cdeghj \sqsubset bghj \sqsubset bfj.$$

Mais $\sqsubset$ *n'est pas total sur* TG_p *en général.*

$T =$



.

$TGM_T = \{c, cd, b, bg, bf, bfj, a\}$. *On a :*

$$c \sqsubset cd \sqsubset b \sqsubset bg \sqsubset bf \sqsubset bfj \sqsubset a.$$

On dit qu'un mot est un facteur-frange de T, si il est facteur de $FRANGE_T$.

Définition 7.19 (Transversale gauche partielle associée à un facteur-frange : tgf)
Le mot u *est une tgf de* T *si :*
(1) u *est une transversale gauche partielle de* T
(2) $FT(u)$ *est un facteur-frange.*

Soit TGF_f est l'ensemble des tgf associées au facteur-frange f. On étend l'ordre $\sqsubset$ aux tgf. Ici encore $min(TGF_f)$ est unique.

Définition 7.20 *Soit l'application* $\vartheta : TGF_f \rightarrow TGF_f$ *définie par :*
(1) si $t = min(TGF_f)$, $\vartheta(t) = t$,
sinon posons $t = t_1 ... t_k$ *et* $P_1 = PERE(t_1)$,
(2) si $\exists i$ *tq.* $t_1 ... t_i = FILS(P_1)$, *alors* $\vartheta(t) = P_1 t_{i+1} ... t_k$,
(3) sinon $\vartheta(t) = t_1 . \vartheta(t_2 ... t_k)$.

Remarque 7.21 *Si* $t \neq min(TGF_f)$, $\vartheta(t) \sqsubset t$. *Comme* TGF_f *possède un minimum unique, ainsi :*

$$\forall t \in TGF_f, \lim_n \vartheta^n(t) = min(TGF_f)$$

Définition 7.22 *Soit l'application* $\theta : TG_p \rightarrow TG_p$ *définie par :*
(1) si $t = tgm_p$, $\theta(t) = t$,
sinon posons $t = t_1 ... t_k$ *et* $P_1 = PERE(t_1)$,
(2) si $\exists i$ *tq.* $t_1 ... t_i = FILS(P_1)$, *alors* $\theta(t) = P_1 t_{i+1} ... t_k$,
(3) sinon $\theta(t) = t_1 . \vartheta(t_2 ... t_k)$, *(*$t_2 ... t_k$ *est une tgf et pas nécessairement une tg, d'où l'utilisation de* ϑ*).*

Ici encore, si $t \neq tgm_p$, $\theta(t) \sqsubset t$. Comme TG_p possède un minimum unique : tgm_p, ainsi :

$$\forall t \in TG_p, \lim_n \theta^n(t) = tgm_p$$

Dans l'exemple précédent, $\theta(cdeghjk) = b.\vartheta(ghjk) = bf.\vartheta(jk) = bfi$, et $\theta^2(cdeghjk) = \theta(bfi) = a$.
Soit p un pré-frange de T. La suite $(\theta^n(p))_n$ converge vers tgm_p.
Posons P_T l'ensemble des pré-franges de T.

Définition 7.23 (tgm associée à un pré-frange) *Soit $\Theta : P_T \to TGM_T$ la fonction définie par :*

$$\Theta(p) = \lim_n \theta^n(p)$$

associe, à tout pré-frange p, sa transversale gauche minimale associée : tgm_p.

On a ici une autre définition de la transversale gauche minimale. v est une transversale gauche minimale de T, si il existe un pré-frange u, tel que $v = \Theta(u)$. On a $TGM_T = \{\Theta(u) : u \text{ pré-frange}\}$.

On dit que F est une sous-forêt de T, si F est un sous-graphe de l'arbre T.

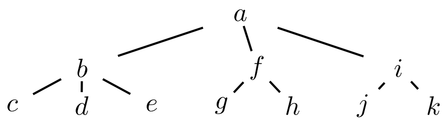
On pose $\emptyset_G$ le graphe vide, qui ne comporte aucun sommet et aucunne arête.

On pose $F(T)$ l'ensemble des sous-forêts de T.

Soit F_1 et F_2 deux sous-forêts de T. $F_3 = F_1 \cup F_2$ si F_3 est tel que : $V_{F_3} = V_{F_1} \cup V_{F_2}$ et $E_{F_3} = E_{F_1} \cup E_{F_2}$.

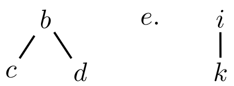
Posons ρ le morphisme de $(TGM(T), .)$ dans $(F(T), \cup)$ définie par : pour tout sommet s de T : $\rho(s)$ est l'arbre dont l'ensemble des sommets est : $\{s, PERE(s)\}$, et dont la seule arête est : $(PERE(s), s)$.
Pour se familiariser avec les concepts, soit l'arbre :

$T =$

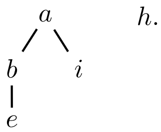


et soit F_1 et F_2 deux sous-forêt de T.

$F_1 =$

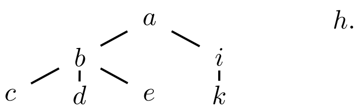


$F_2 =$



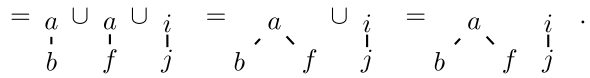
On a

$F_1 \cup F_2 =$



Sur ce même exemple, bfj est une tgm de T. On a :

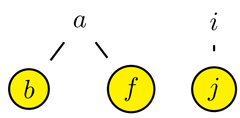
$\rho(bfj) = \rho(b) \cup \rho(f) \cup \rho(j) =$



Toute image de ρ est une suite d'abres de hauteur 1 (sauf $\rho(RACINE_T) =$

$RACINE_T$). On ordonne les sommets de toute image de ρ, suivant l'ordre préfixe des sommets dans T : $\prec_T$. On retrouve l'antécédent de toute image de ρ, en concaténant (suivant $\prec_T$) les feuilles des arbres de hauteur 1.

Par exemple, Soit

$\lambda =$  .

La concaténation (suivant $\prec_T$) des feuilles des arbres de hauteur 1 donne : bfj. En effet, $\rho(bfj) = \lambda$.

Ces remarques impliquent que ρ est injective.

Définition 7.24 (Configuration gauche minimale : cgm) *On appelle l'image de ρ, i.e. $\rho(TGM_T)$, l'ensemble des configurations gauches minimales de l'arbre T, que l'on note CGM_T.*

Remarque 7.25 *Ainsi ρ est une bijection de TGM_T dans CGM_T.*

Remarque 7.26

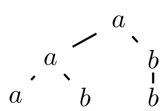
$\rho(RACINE_T) =$  .

7.3 Arbre étiqueté par un morphisme

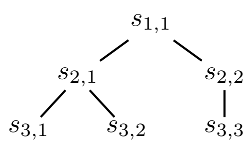
Soit une composition de morphismes appliquée à une lettre : $\varphi_n \circ ... \circ \varphi_1(l)$.

On associe au mot $\varphi_n \circ ... \circ \varphi_1(l)$ un arbre T, étiqueté par un morphisme ξ, tel que :
(1) $V_T = \{s_{i,j} : i, j\}$
(2) $\xi(s_{i,j}) = \varphi_i \circ ... \circ \varphi_1(l)[j]$.

Soient les endomorphismes de $\{a, b\}^*$: h_1 et h_2 vérifiant : $h_1(a) = ab$, $h_2(a) = ab$ et $h_2(b) = b$. On peut représenter le mot $h_2 \circ h_1(a) = aab$ par l'abre suivant :

 .

L'arbre associée à $h_2 \circ h_1(a) = aab$ est (T, ξ) où :

$T =$ 

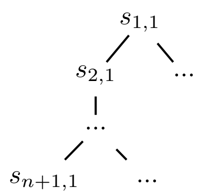
et ξ est telle que :
$\xi(s_{1,1}) = a$,
$\xi(s_{1,1}) = a$, $\xi(s_{1,2}) = b$,
$\xi(s_{1,1}) = a$, $\xi(s_{1,2}) = b$ et $\xi(s_{1,3}) = b$.

7.4 Première démonstration

7.4.1 Etape 1 : arbre étiqueté par un morphisme

On considère l'arbre étiqueté (T, ξ) associé à $\varphi_n \circ .. \circ \varphi_1(l)$ (voir la sous-section 7.3) :

$T =$



et ξ est telle que : $\xi(s_{i,j}) = \varphi_i \circ ... \circ \varphi_1(l)[j]$.

7.4.2 Etape 2 : configurations stables simplifiées

On a une bijection entre les configurations stables de la mémoire 2-pile, et les configurations stables simplifiées (voir la sous-section 7.1).

Par exemple, si l'état de la mémoire de 2-pile est (en notation exponentielle) :

$$\langle abcd\rangle^{\langle abc\rangle a!_1} \langle baaa\rangle^{\langle ba\rangle b\varphi_n!_1} ... \langle ccc\rangle^{\langle cc\rangle c\varphi_n...\varphi_1!_1} Z^{\varphi_n...\varphi_1!_1}$$

alors la mémoire simplifiée est :

$$\varphi_n^{abc} \varphi_{n-1}^{ba} ... \varphi_1^{cc} Z.$$

7.4.3 Etape 3 : confusion entre sommets et étiquettes

Dans les configurations simplifiées, on appréhende les lettre fils de φ_i comme images de sommets de T par le morphisme étiquette ξ.

Par exemple, soit une configuration stable simplifiée de la mémoire 2-pile :

$$\varphi_n^{abc} \varphi_{n-1}^{ba} ... \varphi_1^{cc} Z.$$

Les fils de φ_n constituent (par concaténation suivant l'ordre $\prec_T$) le mot abc. On appréhendera le mot abc comme $\xi(s_{n+1,t_{n+1}} s_{n+1,t'_{n+1}} s_{n+1,t''_{n+1}})$ (pour certains $t_{n+1} < t'_{n+1} < t''_{n+1}$).
De manière génŕale, si les fils de φ_i constituent le mot $u_1...u_j$; alors on appréhende le mot $u_1...u_j$ comme $\xi(s_{i+1,t_{i+1}} ... s_{i+1,t^{(j)}_{i+1}})$.

On n'écrit pas le morphisme ξ, en confondant les sommets avec leurs images par ξ.

Par exemple,

$$h_1^{\xi(s_{2,1})} Z$$

devient :

$$h_1^{s_{2,1}} Z.$$

7.4.4 Etape 4 : bijection entre pré-frange et configuration gauche minimale associée

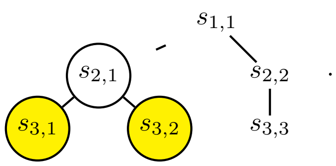
Par Θ (voir la définition 7.23), on a une bijection entre les pré-frange et les transversales gauches minimales.

Par ρ (voir la remarque 7.25), on a une bijection entre les transversales gauches minimales et les configurations gauches minimales.

On a ainsi une bijection entre les pré-franges et les configurations gauches minimales.

Exemple 7.27 *Considérons*

$T =$



Au pré-frange $s_{3,1}.s_{3,2}$*, on associe la transversale gauche minimale* $s_{2,1}$*.*

A la transversale gauche minimale $s_{2,1}$*, on associe la configuration gauche minimale :*

$$\begin{array}{c} s_{2,1} \\ | \\ s_{1,1} \end{array} \quad .$$

Notons que contrairement aux préliminaires combinatoires à la première démonstration, on place le père en bas et le fils en haut.

7.4.5 Etape 5 : bijection entre configurations gauches minimales et configurations stables simplifiées de l'automate

Soit une configuration gauche minimale :

$$\begin{array}{cccc} ... & ... & & ... \\ | & | & & | \\ s_{i_1,l_1} & s_{i_2,l_2} & \cdots & s_{i_k,l_k} \end{array}$$

On a $i_1 < i_2 < ... < i_k$. On associe aux s_{i_j,l_j}, les φ_{i_j}. On a :

$$\begin{array}{cccc} ... & ... & & ... \\ | & | & & | \\ \varphi_{i_1} & \varphi_{i_2} & \cdots & \varphi_{i_k} \end{array}$$

En renversant l'ordre, on a :

$$\begin{array}{cccc} ... & & ... & ... \\ | & & | & | \\ \varphi_{i_k} & \cdots & \varphi_{i_2} & \varphi_{i_1} \end{array}$$

On complète par les arbres à 1 seul sommet : φ_{i_1-1},...,φ_1 et Z. On a :

$$\begin{array}{cccccccc} ... & & ... & ... & & & & \\ | & & | & | & & & & \\ \varphi_{i_k} & \cdots & \varphi_{i_2} & \varphi_{i_1} & .\varphi_{i_1-1} & \cdots & .\varphi_1 & .Z \end{array} \quad .$$

Exemple 7.28 *Soit la configuration gauche minimale :*

$$\begin{array}{c} s_{2,1} \\ | \\ s_{1,1} \end{array}$$

on lui associe la configuration simplifiée :

$$\begin{array}{cc} s_{2,1} & \\ | & \\ h_1 & Z \end{array} \quad .$$

On a défini (informellement) une bijection entre les configurations gauches minimales de T, et les configurations stables simplifiées de l'automate.

7.4.6 Etape 6 : bijection entre pré-franges et configuration stables simplifiée

Par les étapes 3 et 4, on a une bijection entre les pré-franges et les configurations stables simplifiées de l'automate.

7.4.7 Etape 7 : Conclusion

L'étape 6 assure que tout mot en cours de lecture est nécessairement un pré-frange (plus exactement l'image d'un pré-frange par ξ). En effet, si l'automate est en configuration stable, i.e. si il attend la lecture d'une lettre ou si il a consommé entièrement la bande de lecture, alors le mot consommé est un pré-frange, et réciproquement.

Deux cas se présentent, le mot consommé est $FRANGE_T$, ou le mot consommé est un pré-frange propre, i.e. non égal à $FRANGE_T$.

Si $FRANGE_T$ est entièrement lu (plus exactement l'image de $FRANGE_T$ par ξ), alors il est accepté par l'automate. En effet, la configuration stable simplifiée de l'automate correspondante à $FRANGE_T$ est (voir la remarque 7.26) :

$$\begin{array}{c} s_{1,1} \\ | \\ Z \end{array} \quad .$$

En considérant le groupe de transitions K, le mot est accepté.

Si le mot passé à la bande de lecture n'est pas $FRANGE_T$ (c'est néanmoins un pré-frange), alors on vérifie que l'automate ne l'accepte pas. En effet, une fois le mot entièrement consommé, on attend, après le plus d'epsilon-transition possibles, la lecture d'une lettre, or la bande de lecture est vide.

8 Deuxième démonstration

8.1 Préliminaire combinatoire à la deuxième démonstration

Considérons un arbre T (avec racine implicite) parfaitement équilibré.

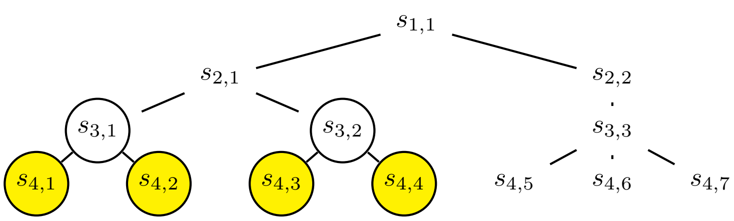
Définition 8.1 (Pré-frange déscendant d'un sommet) *Soit p un pré-frange. p est déscendant du sommet s, si p est égale aux feuilles tombantes de s, i.e. $p = FT(s)$.*

On pose $p_{i,j}$ le pré-frange déscendant de $s_{i,j}$, i.e. $p_{i,j} = FT(s_{i,j})$.

Les sommets de niveaux i de l'arbre sont : $s_{i,1}, ..., s_{i,k_i}$, les pré-frange descendant correspondants sont : $p_{i,1}, ..., p_{i,k_i}$. Si l'arbre est de taille n, alors : $p_{n+1,j} = s_{n+1,j}$ est une feuille.

Exemple 8.2

$T =$



Le shémas illustre : $p_{3,1} = s_{4,1}\ s_{4,2}$ *et* $p_{3,2} = s_{4,3}\ s_{4,4}$.
On a aussi : $p_{2,1} = s_{4,1}\ s_{4,2}\ s_{4,3}\ s_{4,4}$
et : $p_{1,1} = s_{4,1}\ s_{4,2}\ s_{4,3}\ s_{4,4}\ s_{4,5}\ s_{4,6}\ s_{4,7} = FRANGE_T$.

Remarque 8.3 $\exists j', j'' \geq j$ *tel que :* $p_{i,j} = p_{i+1,j'}...p_{i+1,j''}$ *; en reprenant l'exemple précédent, on a par exemple :*

$$p_{2,1} = p_{3,1}\ p_{3,2}$$

et

$$p_{2,2} = p_{3,3} = p_{4,5}\ p_{4,6}\ p_{4,7}.$$

Remarque 8.4

$$p_{n+1,1}...p_{n+1,k_{n+1}} = p_{n,1}...p_{n,k_n} = ... = p_{i,1}...p_{i,k_i} = ... = p_{2,1}...p_{2,k_2} = p_{1,1} = FRANGE_T.$$

8.2 Notations simplifiées

Si T est l'arbre étiqueté par un morphisme associè au mot $\varphi_n \circ ... \circ \varphi_1(l)$, alors on confond les sommets avec les étiquettes.

Dans la sous-section 7.1, on a une simplification des configurations stables de la mémoire 2-pile. On étend cette simplification à n'importe quelle configuration de la mémoire 2-pile (pas seulement les configurations stables). Par exemple, au lieu d'écrire :

$$\varphi_n^{!_1}...\varphi_i^{\varphi_{i+1}...\varphi_n!_1}...\varphi_1^{\varphi_2...\varphi_n!_1}Z^{\varphi_1...\varphi_n!_1}$$

on écrira :

$$\varphi_n...\varphi_i...\varphi_1 Z,$$

et au lieu d'écrire :

$$\varphi^{abcd}$$

on écrira :

$$\varphi^{dcba}.$$

8.3 Nouvelle notation pour les configurations

Les configurations étant ici particulièrement difficiles à lire, on adopte une nouvelle notation. Si q est l'état courant, w le mot restant à lire sur la bande de lecture, et π la mémoire k-pile courante, la notation utilisée jusqu'à présent, pour les configurations de l'automate, peut se résumer comme suit :

$$(q, w, \pi).$$

La notation que l'on utilise ici est :

$$[\pi]_q^w.$$

Elle met l'accent sur l'état de la mémoire k-pile. Par exemple, la configuration :

$$(q, abcd, A_1^{A_2B_2}B_1^{C_2}B_1^{C_2C_2}A_1^{C_1B_1A_1}Z^{A_1B_1B_1})$$

se notera :

$$[A_1^{A_2B_2}B_1^{C_2}B_1^{C_2C_2}A_1^{C_1B_1A_1}Z^{A_1B_1B_1}]_q^{abcd} \quad .$$

8.4 Deuxième démonstration

Dans un premier temps, on démontre, par une récurrence descendante, le lemme ci-dessous.

Lemme 8.5 *Supposons que le mot $\varphi_1...\varphi_n\#$ a été lu, alors pour tout i, la propriété $\mathcal{P}(i)$, définie ci-après, est vraie :*

$$\mathcal{P}(i) : [\varphi_n...\varphi_{i-1}^{w_{i-1}}\varphi_{i-2}^{w_{i-2}}...\varphi_1^{w_1}Z^{w_0}]_{q_2}^{p_{i,j}.w} \quad \vdash_{\mathcal{A}}^* \quad [\varphi_{i-1}^{w_{i-1}.s_{i,j}}\varphi_{i-2}^{w_{i-2}}...\varphi_1^{w_1}Z^{w_0}]_{q_3}^{w} \quad ,$$

où $w, w_{i-1}, w_{i-2}, ..., w_1, w_0$ sont des mots quelconques.
Et les seuls mots acceptés, par dépilement des $\varphi_n...\varphi_i$ sont les $p_{i,j}$.

Démonstration (Récurrence descendante)

– Cas initial : $\mathcal{P}(n)$ est vraie.

On a : $p_{n,j} = p_{n+1,j'}...p_{n+1,j''} = s_{n+1,j'}..s_{n+1,j''}$.
Informellement, on lit les symboles $s_{n+1,j'}..s_{n+1,j''-1}$. A la lecture de chaque symbole $s_{n+1,\lambda}$, on modifie la configuration de la mémoire 2-pile, puis on teste l'égalité aux check-point entre tête de pile de niveau 1 et tête de pile de niveau 2. L'égalité est à chaque fois fausse, on descend jusqu'au plancher, puis on remonte et on lit un nouveau symbole.

Ensuite, On lit symbole $s_{n+1,j''}$, on modifie la configuration de la mémoire en conséquence, et on arrive à un *check-point*, i.e. que l'on va tester l'égalité entre tête de pile de niveau 1 et tête de pile de niveau 2. L'égalité est vraie, on modifie la 2-pile en conséquence. Formellement, on utilise les groupes de transitions E, F, G, H, I et J. On a :

$$[\varphi_n\varphi_{n-1}^{w_{n-1}}...\varphi_1^{w_1}Z^{w_0}]_{q_2}^{s_{n+1,j'}..s_{n+1,j''}.w} \quad \vdash_{\mathcal{A}}^{*} \quad [\varphi_n^{s_{n+1,j'}...s_{n+1,j''}}\varphi_{n-1}^{w_{n-1}}...\varphi_1^{w_1}Z^{w_0}]_{q_4}^{w}.$$

On est à un check-point et l'égalité est vraie : $\varphi_n(s_{n+1,j'}) = s_{n+1,j'}..s_{n+1,j''}$. Par le groupe de transition J (arrivée à un *check-point* : test d'égalité), on a :

$$[\varphi_n^{s_{n+1,j'}..s_{n+1,j''}}\varphi_{n-1}^{w_{n-1}}...\varphi_1^{w_1}Z^{w_0}]_{q_4}^{w} \quad \vdash_{\mathcal{A}}^{*} \quad [\varphi_{n-1}^{w_{n-1}.s_{n+1,j'}}...\varphi_1^{w_1}Z^{w_0}]_{q_3}^{w},$$

or $s_{n+1,j'} = s_{n,j}$ (dû au fait que les morphismes sont de première lettre). En outre, les seuls mots acceptés, par dépilement de φ_n, sont de la forme : $p_{n,j}$.

– Supposons que $\forall p \geq i+1$, $\mathcal{P}(p)$ est vraie.

– On montre que $\mathcal{P}(i)$ est vraie.

On a : $p_{i,j} = p_{i+1,j'}...p_{i+1,j''}$.

Par l'hypothèse de récurrence,

$$[\varphi_n...\varphi_i\varphi_{i-1}^{w_{i-1}}\varphi_{i-2}^{w_{i-2}}...\varphi_1^{w_1}Z^{w_0}]_{q_2}^{p_{i+1,j'}...p_{i+1,j''}.w} \quad \vdash_{\mathcal{A}}^{*} \quad [\varphi_i^{s_{i+1,j'}...s_{i+1,j''}}\varphi_{i-1}^{w_{i-1}}\varphi_{i-2}^{w_{i-2}}...\varphi_1^{w_1}Z^{w_0}]_{q_3}^{w}.$$

Avec le groupe de transition I, on modifie la configuration de la mémoire 2-pile (avec les notations simplifiées, on ne voit pas les changements). On se trouve maintenant à un *check-point* (marqué par l'état q_4). L'égalité entre tête de pile de niveau 1 et tête de pile de niveau 2 est vraie : $\varphi_n(s_{n+1,j'}) = s_{n+1,j'}..s_{n+1,j''}$. Par le groupe de transition J (arrivée à un *check-point* : test d'égalité), on a :

$$[\varphi_i^{s_{i+1,j'}...s_{i,j''}}\varphi_{i-1}^{w_i}\varphi_{i-2}^{w_{i-2}}...\varphi_1^{w_1}Z^{w_0}]_{q_4}^{w} \quad \vdash_{\mathcal{A}}^{*} \quad [\varphi_{i-1}^{w_i.s_{i+1,j'}}\varphi_{i-2}^{w_{i-2}}...\varphi_1^{w_1}Z^{w_0}]_{q_3}^{w}.$$

Or $s_{i+1,j'} = s_{i,j}$ (dû au fait que les morphismes sont de première lettre). En outre, par hypothèse de récurrence, les seuls mots acceptés par dépilement des $\varphi_n...\varphi_{i+1}$ sont les $p_{i+1,j}$. Ici, si le mot n'était pas de la forme $p_{i,j}$, le test d'égalité au *check-point* serait faux ; on continuerait à lire jusqu'au moment où le symbole de tête de pile de niveau 1 n'appartiendrait plus à l'alphabet de la 2-pile.

■

Remarque 8.6

$$\mathcal{P}(1) : \qquad [\varphi_n...\varphi_1Z^{w_0}]_{q_2}^{p_{1,j}.w} \quad \vdash_{\mathcal{A}}^{*} \quad [Z^{w_0.s_{1,j}}]_{q_3}^{w} \quad ,$$

et les seul mot acceptés, par dépilement des $\varphi_n...\varphi_1$ sont les $p_{1,j}$. Notons, toutefois, que le seul mot de la forme $p_{1,j}$ est $p_{1,1}$.

On démontre maintenant, que l'automate $\mathcal{A}$, reconnaît l'image de la composition de morphismes $\varphi_n \circ ... \circ \varphi_1$, sur toute lettre l de A.

Par les groupes des transitions A, B, C, D et E :

$$[Z]_{q_0}^{\varphi_1 ... \varphi_n \# p_{1,1}} \quad \vdash_{\mathcal{A}}^{*} \quad [\varphi_n ... \varphi_1 Z]_{q_2}^{p_{1,1}}.$$

En utilisant le lemme 8.5 (plus particulièrement la remarque 8.6) :

$$[\varphi_n ... \varphi_1 Z]_{q_2}^{p_{1,1}} \quad \vdash_{\mathcal{A}}^{*} \quad [Z^{s_{1,1}}]_{q_0}^{\epsilon}$$

et le seul mot accepté, par dépilement des $\varphi_n ... \varphi_1$ est $p_{1,1}$.

Par le groupe de transitions K :

$$[Z^{s_{1,1}}]_{q_0}^{\epsilon} \quad \vdash_{\mathcal{A}}^{*} \quad [\epsilon]_{q_0}^{\epsilon}.$$

Ainsi le seul mots accepté par l'automate, après lecture de $\varphi_1 ... \varphi_n \#$, est $p_{1,1} = \varphi_n \circ ... \circ \varphi_1(l)$.

Cela démontre : $L_1 = L(\mathcal{A})$.

9 Rationnel

Proposition 9.1 *Si X est un ensemble fini d'endomorphismes de première lettre, et $R \in Rat(X^*)$, alors $L_2 = \{w(A) : w \in R\}$ est un langage d'index.*

Démonstration
Par hypothèse, R est rationnel. Considérons l'automate fini reconnaissant R. On le note $\mathcal{A}_R$. Considérons l'automate 2-pile reconnaissant L_1. On le note $\mathcal{A}_{L_1}$. On construit un automate 2-pile $\mathcal{A}_{L_2}$ qui reconnaît L_2. On explique son fonctionnement de manière informelle. On distingue 3 phases dans l'exécution de $\mathcal{A}_{L_2}$.

• **Phase 1 : Initialisation**
Initialement, la mémoire 2-pile est :

$$Z$$

et on ne lit pas encore la bande de lecture. On empile $!_1$ au niveau 1 ; la mémoire est :

$$Z^{!_1}$$

puis on passe à la phase 2.

• **Phase 2 : génération de $w \in R$**
On ne commence pas encore à lire la bande de lecture. Par hypothèse, $w = w_n...w_1 \in R$ est reconnu par $\mathcal{A}_R$.

La partie 2 de $\mathcal{A}_{L_2}$ est basée sur $\mathcal{A}_R$.

Dès qu'un w_i est donné par $\mathcal{A}_R$, on empile w_i au niveau 1 (plus exactement, on empile le symbole de pile de niveau 1 qui représente w_i).

A la fin de cette partie, la mémoire 2-pile est :

$$Z^{w_1...w_n!_1}$$

puis on passe à la phase 3.

• **Phase 3 : reconnaissance de $w(x)$**
On commence à consommer la bande de lecture. La suite est identique à l'automate $\mathcal{A}_{L_1}$, une fois le symbole # consommé.

■

Remarque 9.2 *De plus, si R est déterministe (resp. non déterministe), alors L_3 est déterministe (resp. non déterministe).*

10 Rationnel et endomorphisme partiellement de première lettre

Soit φ un endomorphisme partiellement de première lettre de A^*. Posons E le plus grand sous-ensemble de A, dont tous les éléments on pour image ϵ par φ. On a :
(1) $\varphi(E) = \{\epsilon\}$
(2) $\varphi(E') = \{\epsilon\} \Rightarrow E' \subset E$.

Considérons l'ensemble $\widetilde{E} = \{\widetilde{x} : x \in E\}$, équipotent à E et disjoint de A.

On définit l'homomorphisme $\widetilde{\varphi} : A^* \to (A \cup \widetilde{E})^*$ qui vérifie :
(1) si $x \in E$, $\widetilde{\varphi}(x) = \tilde{x}$
(2) sinon $\widetilde{\varphi}(x) = \varphi(x)$.

On a : $\widetilde{\varphi}(E) = \widetilde{E}$.

Définition 10.1 (Automate $\mathcal{A}_{\widetilde{L_1}}$) *$\mathcal{A}_{\widetilde{L_1}}$ correpsond à l'automate $\mathcal{A}_{L_1}$ qui reconnaît L_1 (voir section ?), auquel on adjoint de nouveaux symboles de pile, ainsi que des codes. Pour tout x de E, on considère le symbole de niveau 1 : $\widetilde{x}_1$ et le symbole de niveau 2 : $\widetilde{x}_2$. Si φ^1 (resp. φ^2) est le symbole de 1-pile (resp. 2-pile) représentant φ, alors on code $\varphi^1(E) = \{\widetilde{x}_1 : x \in E\}$ et $\varphi^2(E) = \{\widetilde{x}_2 : x \in E\}$.*

Lemme 10.2 *Si X est ensemble fini d'endomorphismes partiellement de première lettre, alors*

$$\widetilde{L_1} = \{w\#\widetilde{w}(A) : w \in X^*\}$$

est un langage d'index déterministe,

Démonstration En effet, l'automate $\mathcal{A}_{\widetilde{L_1}}$ reconnaît $\widetilde{L_1}$.
■

Lemme 10.3 *Si X est un ensemble fini d'endomorphismes partiellement de première lettre, et $R \in Rat(X^*)$, alors $\widetilde{L_2} = \{\widetilde{w}(A) : w \in R\}$ est un langage d'index.*

Remarque 10.4 *De plus, si R est deterministe (resp. non deterministe), alors L_3 est deterministe (resp. non deterministe).*

Démonstration Identique à la démonstration de de la proposition 9.1.
■

Théorème 10.5 (Résultat principal de la deuxième partie du mémoire) *Si X est un ensemble fini d'endomorphismes partiellement de première lettre, et $R \in Rat(X^*)$, alors $L_3 = \{w(A) : w \in R\}$ est un langage d'index.*

Remarque 10.6 *On perd ici le déterminisme.*

Démonstration Considérons l'homorphisme $\mu : (A \cup \widetilde{E})^* \to A^*$ définit par :
(1) si $x \in A$, $\mu(x) = x$
(2) si $x \in \widetilde{E}$, $\mu(x) = \epsilon$.

Alors, on a : $\mu(\widetilde{L_2}) = L_3$.

Les langages de niveau k (non déterministes) étant clos par homomorphisme, on conclut que L_3 est un langage d'index (mais on perd le déterministe).

■

Quatrième partie

Conclusion et perspectives

11 Conclusion : structure des solutions-mot

Dans la deuxième partie du mémoire, on a démontré que :

l'ensemble des solutions-morphisme de toute équation, ou système d'équations, en mot quadratique est un rationnel.

Dans la troisième partie du mémoire, on a démontré que :

si X est un ensemble fini d'endomorphismes partiellement de première lettre, et $R \in Rat(X^*)$, alors $L_3 = \{w(A) : w \in R\}$ est un langage d'index,

où A est un alphabet.

On remarque que les morphismes-étiquette de l'automate, qui reconnaît les solutions-morphisme de toute équation, tout système d'équations, en mot quadratique, sont des endomorphismes de première lettre.

Ainsi l'ensemble des solutions-morphisme de toute équation, ou système d'équations, en mots quadtratique est un rationnel d'un ensemble fini d'endomorphismes partiellement de première lettre.

En utilisant le résultat de la troisième partie, on trouve :

$$\{\sigma(x) : \sigma \in Sol_{\mathcal{E}}(E), x \in \mathcal{V}\} \text{ est un langage d'index,}$$

où $\mathcal{V}$ est l'ensemble des variables de l'équation, ou du système d'équations, en mot quadratique E.

Si $v = |\mathcal{V}|$, posons : $\mathcal{V} = \{x_1, ..., x_v\}$.

On rappelle la bijection $\beta : Sol_{\mathcal{E}}(E) \to Sol_{\mathcal{M}}(E)$:

$$\beta(\sigma) = (\sigma(x_1), ..., \sigma(x_v)).$$

On a ainsi :

$$Sol_{\mathcal{M}}(E) = \{\sigma(x_1)\#...\#\sigma(x_v) : \sigma \in Sol_{\mathcal{E}}(E)\} \text{ est un langage d'index.}$$

Théorème 11.1 (Théorème principal de la quatrième partie du mémoire) *L'ensemble des solutions-mot d'une équation, ou d'un système d'équations, en mot quadratique est un langage d'index.*

12 Perspectives

Il reste à traiter le cas des équations générales, mais sous-ensemble des solutions d'exposant majoré par un entier fixe e.

13 Remerciements

Je remercie Mr. David Renault et Mr. Géraud Sénizergues, pour le sérieux avec lequel ils m'ont encadré.

Références

[Abd90] Habib Abdulrab. Implementation of makanin's algorithm. In *IW-WERT*, pages 61–84, 1990.

[Aho68] Alfred V. Aho. Indexed grammars – an extension of context-free grammars. *J. ACM*, 15(4) :647–671, 1968.

[Aho69] Alfred V. Aho. Nested stack automata. *J. ACM*, 16(3) :383–406, 1969.

[Cau02] Didier Caucal. On infinite terms having a decidable monadic theory. In *MFCS '02 : Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science*, pages 165–176, London, UK, 2002. Springer-Verlag.

[DGH05] Volker Diekert, Claudio Gutierrez, and Christian Hagenah. The existential theory of equations with rational constraints in free groups is pspace-complete. *Inf. Comput.*, 202(2) :105–140, 2005.

[Die02] Volker Diekert. Makanin's algorithm. *In Algebraic Combinatorics on Words, M. Lothaire*, chapter 12 :387–442, 2002. Cambridge University Press, Cambridge.

[DP02] Robert Dąbrowski and Wojtek Plandowski. On word equations in one variable. *Mathematical Foundations of Computer Science 2002*, volume 2420 de Lecture Notes in Comput. Sci. :212–220, 2002. Berlin.

[DP04] Robert Dąbrowski and Wojtek Plandowski. Solving two-variable word equations (extended abstract). *Automata, Languages and Programming*, Volume 3142 de Lecture Notes in Comput. Sci. :408–419, 2004. Berlin.

[Fra05] Severine Fratani. *Automates à pile de piles ... de piles.* PhD thesis, Université Bordeaux 1, 2005.

[FS03] S. Fratani and G. Sénizergues. Iterated pushdown automata and sequences of rational numbers. In *St Petersburg second Logic Days*, 2003. Abstract at : http ://logic.pdmi.ras.ru/2ndDays/index.html ;full article submitted to special issue of Annals of Pure and Applied Logics.

[HL] Markus Holzer and Klaus-Jörn Lange. On the complexities of linear ll(1) and lr(1) grammars.

[KMP00] Juhani Karhumäki, Filippo Mignosi, and Wojciech Plandowski. The expressibility of languages and relations by word equations. *Journal of the ACM*, pages 483–505, 2000.

[Mak77] G. S. Makanin. The problem of solvability of equations in a free semi-group. *Math. USSR Sbornik*, 32(2) :129–198, 1977.

[Mas74] A.N. Maslov. The hierarchy of indexed languages of an arbitrary level. *Soviet Math. Dokl.*, 15, 1974.

[Mas76] A.N. Maslov. Multi-level stack automata. *Probl. of Inf. Transm.*, 12 :38–43, 1976.

[Pla99] Wojciech Plandowski. Satisfiability of word equations with constants is in nexptime. In *STOC '99 : Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 721–725, New York, NY, USA, 1999. ACM Press.

[Pla04] Wojciech Plandowski. Satisfiability of word equations with constants is in pspace. *J. ACM*, 51(3) :483–496, 2004.

[RD99] John Michael Robson and Volker Diekert. On quadratic word equations. *Lecture Notes in Computer Science*, 1563 :217–226, 1999.

[Sch92] Klaus U. Schulz. Makanin's algorithm for word equations - two improvements and a generalization. In *IWWERT '90 : Proceedings of the First International Workshop on Word Equations and Related Topics*, pages 85–150, London, UK, 1992. Springer-Verlag.