Automates à $n - piles$

David Gontier

19 décembre 2008

1 Introduction

Il est classique de montrer que les automates reconnaissent les langages rationnels, et que les automates à piles reconnaissent les grammaires non contextuelles. De plus, si on munit un automate d'une 2^{me} pile, il devient équivalent à une machine de Turing. On se demande alors naturellement ce que pourrait reconnaître un automate à pile, dont les éléments de la pile seraient des piles elle-mêmes. C'est ce qu'on appelle un automate à $n - pile$.

Nous commençerons par définir les automates à $n - pile$ dans le cas général, puis nous nous intéresserons plus spécialement aux automates à $2 - piles$. Nous introduirons les langages polynomiaux, et démontrerons qu'ils sont reconnus par de tels automates. Enfin, nous présenterons le langage de Uryzyczyn, qui en particulier amène à la question encore non résolue de savoir si on peut déterminiser un automate à $n - pile$ pour $n \geq 2$, les cas $n = 0$ et $n = 1$ étant vrais.

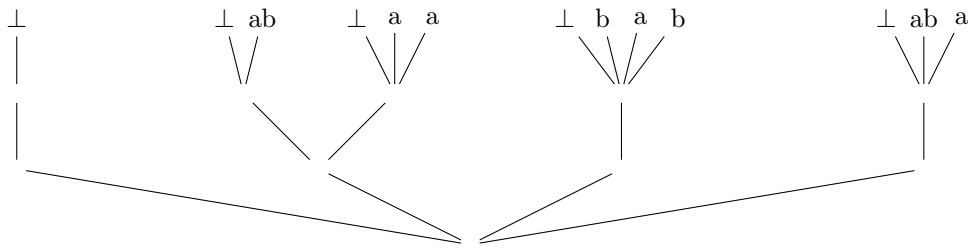
2 Automates à piles de piles

La définition formelle d'un automate à piles de piles étant non seulement complexe à formaliser entièrement, et ne présentant que peu d'intérêt pour cette simple introduction, on passera rapidement sur les définitions, mais on illustrera toute la suite par de nombreux exemples.

Definition 1. On définit récursivement une $n - pile$ de la manière suivante :
- Une $1 - pile$ est une suite finie $[a_1, \dots, a_n]$ de mots de A^*
- Une $(n+1) - pile$ est une suite finie $[p_1, \dots, p_l]$ de $n - piles$

On note $\perp_k$ la $k - pile$ vide

Exemple 2. $[[[\perp]], [[\perp, ab], [\perp, a, a]], [[\perp, b, a, b]], [[\perp, ab, a]]]$ est une $3 - pile$, que l'on peut aussi représenter sous forme d'un arbre :



On s'autorise les opérations suivantes sur les $n-piles$, pour $1 < k < n$:

$$\begin{cases} \text{top } [a_0, \ldots, a_n] & = & a_0 & \text{si } n = 1 \\ \text{top } [p_0, \ldots, p_l] & = & \text{top } p_0 & \\ \text{push}_1(\text{a}) \ [a_0, \ldots, a_n] & = & [a, a_0, \ldots, a_n] & \text{si } n = 1 \\ \text{push}_n \ [p_0, p_1, \ldots, p_l] & = & [p_0, p_0, p_1, \ldots, p_l] & \\ \text{push}_k \ [p_0, p_1, \ldots, p_l] & = & [\text{push}_k p_0, p_1, \ldots, p_l] & \\ \text{pop}_n \ [p_0, p_1, \ldots, p_l] & = & [p_1, \ldots, p_l] & \\ \text{pop}_k \ [p_0, p_1, \ldots, p_l] & = & [\text{pop}_k p_0, p_1, \ldots, p_l] & \end{cases}$$

En particulier, on remarque qu'on ne travaille qu'avec le haut de la pile.

Definition 3. Un automate à $n-pile$ ($n-PDA$) est un automate "muni" d'une $n-pile$ P, et dont les transitions sont de la forme $\delta_\alpha(q, a) = (p, action)$ où top $P = a$, α est le symbole lu par l'automate, et où $action$ est une ou plusieurs des opérations définies précédemment sur P. On travaillera avec le mode d'acceptation par pile vide.

Exemple 4. L'automate déterministe à $2-pile$ suivant reconnait le langage $L = \{a^n b^n c^n, n \geq 0\}$:

$$\begin{cases} \delta_a(q_o, \perp_2) & = & (q_0, push_1(Z)) \\ \delta_a(q_o, Z) & = & (q_0, push_1(Z)) \\ \delta_b(q_0, Z) & = & (q_1, push_2 + pop_1) \\ \delta_b(q_1, Z) & = & (q_1, pop_1) \\ \delta_c(q_1, \perp_1) & = & (q_2, pop_2 + pop_1) \\ \delta_c(q_2, Z) & = & (q_2, pop_1) \end{cases}$$

Dans l'état q_0, on empile n fois Z dans la $1-pile$. Puis on copie cette pile une 2^{me} fois grâce à push_2. Il suffit de vérifier avec ces 2 piles qu'il y a le bon nombre de b et c.

On s'intéresse maintenant plus spécialement aux automates à $2-piles$.

3 Langages polynomiaux

Dans cette partie, on introduit les langages polynomiaux, et on démontre qu'ils sont reconnaissables par des automates à $2-piles$. En plus, des actions

définies précédemment, on introduit des nouvelles notations pour faciliter la compréhension :

$$\begin{cases} \text{pop } [p_0, p_1, \ldots, p_n] & = & \begin{cases} \text{pop}_1 & \text{si } p_0 \neq \perp \\ \text{pop}_2 & \text{sinon} \end{cases} \\ \text{popifnot}(\omega)\ \text{P} & = & \begin{cases} \text{pop} & \text{si top(P)} \neq \omega \\ \text{rien} & \text{sinon} \end{cases} \\ \text{popto}(\omega)\ \text{P} & = & (\text{popifnot}(\omega))^{\infty}\ \text{P} \end{cases}$$

popto vide en fait la pile jusqu'à la pile vide, ou jusqu'à ce que top(P) soit égal à ω.

On notera enfin $\overline{n}$ pour la $1-$*pile* $\underbrace{Z, \ldots, Z}_{n}, \perp$, Z étant un symbole quelconque.

On appellera "pile 1" la $1-$*pile* accessible, et "pile 2" la $2-$*pile* de l'automate considéré.

Definition 5. Soit $\mathcal{A}$ un alphabet, un langage L est dit polynomial sur $\mathcal{A}$ si il existe $(\omega_{i,j})_{1 \leq i \leq m, 1 \leq j \leq n} \in \mathcal{A}$, tel que :

$$L = \left\{ \omega_{1,1}^{k_1} \omega_{1,2}^{k_1^2} \ldots \omega_{1,n}^{k_1^n} \omega_{2,1}^{k_2} \ldots \omega_{m,n}^{k_m^n} | k_1, \ldots, k_m \in \mathbb{N}, \omega_{i,1} \neq \epsilon \right\}$$

Exemple 6. Le langage $\left\{ a^n b^{n^2}, n \in \mathbb{N} \right\}$ est polynomial, le langage $\{a^n b^m c^{nm}, n, m \in \mathbb{N}\}$ ne l'est pas.

Theoreme 7. *Les langages polynomiaux sont reconnus par les 2-PDA.*

On peut tout de suite remarquer que la réciproque est fausse : le langage $\{a^n b^m c^{nm}, n, m \in \mathbb{N}\}$ étant reconnu par l'automate suivant :

- push_1 a la lecture de a
- push_2 a la lecture de b
- pop a la lecture de c

Démonstration. On remarque qu'on peut se restreindre au cas

$$L = \left\{ \omega_1^k \omega_2^{k^2} \ldots \omega_n^{k^n}, k_1, \ldots, k_m \in \mathbb{N}, \omega_1, \ldots, \omega_n \in \mathcal{A}, \omega_1 \neq \epsilon \right\}$$

quitte à mettre plusieurs automates à la suite, ou à ajouter des états.

On suppose de plus que, si i est le premier indice plus grand que 2, alors $\omega_1 \neq \omega_i$. Cela permet d'identifier facilement k lors de la lecture.

La démonstration se fait par récurrence : on va montrer qu'on peut construire un automate qui, à partir de $[\overline{k}]$ peut reconnaitre a^{k^d} par liste vide :

Le cas n=1 est trivial, en posant simplement $\delta_a(q_o, Z) = (q_0, pop)$

Le cas n=2 peut se traiter ainsi : On remarque que $n^2 = \sum_{i=1}^{n} 2i - 1$. On commence par poper, puis ajouter 0,1 et 1 en sommet de pile (C'est le "code" de $2(n-1)^1 + 1(n-1)^0 = n^2 - (n-1)^2$). Puis, si on lit 1, on copie la liste (push_2), on dépile les 1 et 0 restant (popto(Z)), et on reconnaît $n-1$ avec la méthode précédente, puis on recommence (on peut marquer la première pile en

lui ajoutant un symbole distinctif X pour simuler la pile vide : la pile vide de l'hypothèse de récurrence correspond alors à top(P) = X). Si on lit 0, on dépile le 0 (pop) en lisant un a. Sinon, et si la pile n'est pas encore vide, on pop_1 et on recommence tout.

Si on note S_n le nombre de lettres reconnues par cette méthode, on a la relation de récurrence $S_n = S_{n-1} + 2(n-1) + 1$, et $S_0 = 0$ ce qui donne bien $S_n = n^2$.

Les cas suivants se traitent avec la même méthode : on calcule $X^d - (X-1)^d$ qui est un polynome en X de degré (d-1) : $a_0 + \cdots + a_{d-1}X^{d-1}$. On pop une fois, on met le code obtenu : on empile a_0 fois 0,... , a_{d-1} fois (d-1), et suivant ce qu'on lit au dessus, et avec les hypothèses de récurrence, on reconnaît k^d avec la même méthode que pour le cas $n = 2$. □

Exemple 8. Pour $\left\{a^n b^{n^2}, n \in \mathbb{N}\right\}$, la pile aura successivement les états suivants :

$a^n b^{n^2}$	$[[\bot]]$
$a^{n-1} b^{n^2}$	$[[Z\ \bot]]$
b^{n^2}	$[[\overline{n}\ \bot]]$
b^{n^2}	$[[110\overline{(n-1)}\ \bot]]$
b^{n^2}	$[[X10\overline{(n-1)}\ \bot][X10\overline{(n-1)}\ \bot]]$
b^{n^2}	$[[\overline{(n-1)}\ \bot][X10\overline{(n-1)}\ \bot]]$
b^{n^2-n+1}	$[[\bot][X10\overline{(n-1)}\ \bot]]$
b^{n^2-n+1}	$[[X0\overline{(n-1)}\ \bot][X0\overline{(n-1)}\ \bot]]$
b^{n^2-n+1}	$[[\overline{(n-1)}\ \bot][X0\overline{(n-1)}\ \bot]]$
b^{n^2-2n+2}	$[[\bot][X0\overline{(n-1)}\ \bot]]$
$b^{(n-1)^2}$	$[[\overline{(n-2)}\ \bot]]$
$\vdots$	$\vdots$

4 Langage de Urzyczyn

Soit le problème suivant : Un enfant reçoit dans l'ordre des objets numérotés de 1 à n, qu'il met dans une pile. Cet enfant retire de temps et temps pendant le processus le premier objet du sommet de la pile, et le jette à la poubelle. Le problème est de savoir quel est le numéro du jouet du dessus de la pile à la fin de l'opération. On se demande si on peut résoudre ce problème avec un automate à $n - pile$.

Formalisons un peu ce problème :

Definition 9. Le langage U de Urzyczyn est défini sur l'alphabet $\mathcal{A} = \{(,), *\}$ et consiste en l'ensemble des mots qui s'ecrivent sous la forme $w*^n$, tel que w soit le préfixe d'une expression bien parenthésée, et qu'aucun préfixe de w soit bien paranthésé. De plus, on donne une étiquette à chaque parenthèse de la manière suivante :

– L'étiquette du n^{eme} (est n

– L'étiquette de) est celle de l'étiquette de la parenthèse précédant celle du (qu'elle ferme.
– n est alors le numéro de la dernière étiquette

Exemple 10.

$$\underset{1}{(}\ \underset{2}{(}\ \underset{3}{(}\ \underset{4}{)}\ \underset{3}{)}\ \underset{2}{(}\ \underset{5}{(}\ \underset{6}{)}\ \underset{5}{(}\ \underset{7}{(}\ \underset{8}{)}\ \underset{7}{)}\ \underset{5}{)}\ * \ * \ * \ * \ *$$

En terme du problème, la parenthèse $\underset{n}{(}$ représente l'insertion de l'objet n dans la pile, et la parenthèse $\underset{n}{)}$ représente l'action de retirer un objet, n étant toujours le numéro de l'objet en sommet de pile après l'action. Le nombre final de * est juste une manière de coder le numéro de l'objet en sommet de pile à la fin de l'opération.

Proposition 11. *Tout mot de U s'écrit de manière unique sous la forme :*

$$\underbrace{(\underbrace{(\ldots}_{(1)}\underbrace{(\ldots)\ldots(\ldots)}_{(2)}\underbrace{)}*\cdots*}_{(3)}$$

où :
– *(1) est le préfixe d'un mot bien parenthésé, finissant par (, dont aucun sous préfixe n'est bien paranthésé.*
– *(2) est une expression bien paranthésée.*
– *(3) a le même nombre n de * que le nombre de (dans (1).*

Exemple 12. Le mot $((((())(()(()) * * * * *$ admet la décomposition suivante :

$$\underbrace{(\ (\ (\ (\ (\ )\ )\ (\ )\ (\ (\ )\ )}_{(1)\qquad\qquad(2)}\underbrace{*\ *\ *\ *\ *}_{(3)}$$

Démonstration. La proposition est triviale quand on réfléchit en terme du problème : n correspond simplement au numéro de l'objet au sommet de la pile de l'enfant à la fin de l'opération. □

Réciproquement, on vérifie aisément que l'ensemble des mots ayant une telle décomposition sont dans U.

Theoreme 13. *Le langage U est reconnaissable par un automate à $2-$pile non déterministe.*

Démonstration. L'idée est de deviner la partie (1) du mot que l'on reconnaît : On va vérifier d'une part que le mot qu'on lit est bien paranthésé, grâce à la pile 1, et la trace du nombre de (qu'on lit dans la pile 2 : dans la partie (1) du mot, le lecture de (provoque un push_1 suivi d'un push_2(sauf le premier), la lecture d'un) provoque juste pop_1. Dans la partie (2), on vérifie juste que le mot est bien paranthésé : on crée une nouvelle pile de travail : push_2 puis popto($perp). La lecture de (fait push_1, et la lecture de) fait pop_1. Enfin, la lecture de * fait deux pop_2 au début, puis un pop_2 pour chaque * lu supplémentaire. La pile à la fin doit être vide. □

Exemple 14. Dans l'exemple précèdent, on obtient les piles suivantes :

$$
\begin{aligned}
& [\bot] \\
\xrightarrow{(} \quad & [[Z\ \bot]] \\
\xrightarrow{(} \quad & [[ZZ\ \bot], [Z\ \bot]] \\
\xrightarrow{(} \quad & [[ZZZ\ \bot], [ZZ\ \bot], [Z\ \bot]] \\
\xrightarrow{(} \quad & [[ZZZZ\ \bot][ZZZ\ \bot], [ZZ\ \bot], [Z\ \bot]] \\
\xrightarrow{)} \quad & [[ZZZ\ \bot][ZZZZ\ \bot], [ZZ\ \bot], [Z\ \bot]] \\
\xrightarrow{)} \quad & [[ZZ\ \bot][ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot],] \\
\xrightarrow{(} \quad & [[ZZZ\ \bot][ZZZ\ \bot][ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot]] \\
\xrightarrow{\epsilon} \quad & [[\bot][ZZZ\ \bot][ZZZ\ \bot][ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot]] \\
\xrightarrow{(} \quad & [[Z\ \bot][ZZZ\ \bot][ZZZ\ \bot][ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot]] \\
\xrightarrow{)} \quad & [[\bot][ZZZ\ \bot][ZZZ\ \bot][ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot]] \\
\xrightarrow{(} \quad & [[Z\ \bot][ZZZ\ \bot][ZZZ\ \bot][ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot] \\
\xrightarrow{(} \quad & [[ZZ\ \bot][ZZZ\ \bot][ZZZ\ \bot][ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot]] \\
\xrightarrow{)} \quad & [[Z\ \bot][ZZZ\ \bot][ZZZ\ \bot][ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot]] \\
\xrightarrow{)} \quad & [[\bot][ZZZ\ \bot][ZZZ\ \bot][ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot]] \\
\xrightarrow{*} \quad & [[ZZZ\ \bot][ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot]] \\
\xrightarrow{*} \quad & [[ZZZZ\ \bot], [ZZZ\ \bot], [ZZ\ \bot]] \\
\xrightarrow{*} \quad & [[ZZZ\ \bot], [ZZ\ \bot]] \\
\xrightarrow{*} \quad & [[ZZ\ \bot]] \\
\xrightarrow{*} \quad & [\bot_2]
\end{aligned}
$$

Actuellement, on ne sait toujours pas si le langage U est reconnaissable par un automate à $n - pile$ déterministe.

5 Conclusion

Nous avons vu des exemples de langage reconnus par des 2-PDA sans toutefois avoir réussi à caractériser complètement la classe des langages qu'ils reconnaissent ($LANG_2$). On peut trouver une caractérisation de ce langage dans [3] et [4]. On peut de même se poser la question pour des piles d'ordre supérieur ($LANG_n$). En pratique, on obtient des langages intéressants (grammaire récursive permettant la vérification de langages fonctionnels) en ajoutant une propriété "d'effondrement" à ces automates. On pourra se reporter à [3] ou [5] pour plus de détails.

Références

[1] JOLIE G. DE MIRANDA *: Structures generated by higher-order grammars and the safety constraint,* Thèse Trinity Term, 2006

[2] W. KARIANTO *: On Parikh images of higher-order pushdown automaton (extended abstract)*

[3] S. FRATANI *: Automates à piles de piles ...de piles,* Thèse Université Bordeaux 1, 2005

[4] N. MARIN *: Suites de mots et automates,* Université Bordeaux 1, 2007

[5] T. KNAPIK, D. NIWINSKI, P. URZYCZYN *: Higher-order pushdown trees are easy,* Université de la Réunion, 2002