Langages formels (11X003) - Automne 2024

7. Automates à pile

Enseignant: Arnaud Casteigts

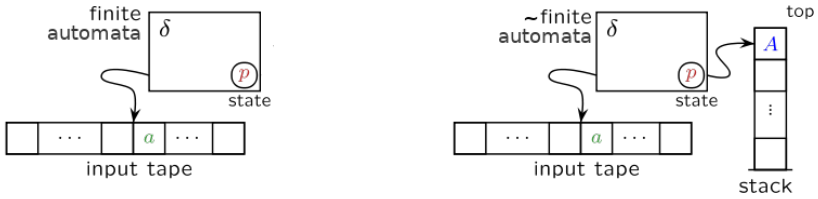
Assistants: A.-Q. Berger & M. De Francesco
Monitrices: L. Heiniger & A. Tekkoyun

(L'équivalence entre les grammaires régulières et les langages réguliers a été ajoutée aux notes de la semaine dernière, Cours 6 : Grammaires formelles, Section 6.5).

Nous avons vu que certains langages ne sont pas réguliers. Ils ne peuvent donc pas être reconnus par des automates finis (AFD ou AFN, c'est pareil). Certains de ces langages, comme le langage $L = \{a^n b^n \mid n \in \mathbb{N}\}$, peuvent cependant être décrits par une grammaire hors-contexte. Nous allons voir aujourd'hui un modèle de machine plus puissant, qui est capable de reconnaître tous les langages hors-contextes. Ce modèle est celui des *automates à pile*.

7.1 Description intuitive

Si l'on dessine ce qu'est un automate fini (AF, pour rappel), il y a la partie avec les états et les transitions, mais aussi la partie sur laquelle on lit le mot en entrée (souvent non dessinée). Cette partie est appelée la **bande** (historiquement, les informations étaient stockées sur des bandes magnétiques), ou *tape*, en anglais. L'AF utilise implicitement une *tête de lecture* qui pointe sur le symbole suivant sur la bande (Figure 1a). Comme déjà discuté, la principale limitation des AFs est qu'ils n'ont pas de mémoire. Les automates à pile (AP) consistent à augmenter ces machines avec mémoire potentiellement infinie, en l'occurrence, une **pile** (ou *stack*, en anglais). Cette dernière permet à l'automate de stocker de l'information en **empilant** et **dépilant** des éléments lorsqu'il effectue ses transitions. À tout moment, seul le **sommet** de la pile (dernier élément empilé) est accessible (voir la Section 7.5 si vous ne connaissez pas les piles en général).



(a) Automate fini (b) Automate à pile

FIGURE 1 – Représentation d'automates finis ou à pile.

Mis à part la pile, les APs fonctionnent comme des AFs. Il existe une version déterministe (APD) et non-déterministe (APN). Contrairement aux automates finis, les APDs reconnaissent strictement moins de langages que les APNs (le non-déterminisme augmente réellement les capacités). On appelle langages hors contexte *déterministes* les langages reconnaissables par des APDs (quelque part entre les langages réguliers et les langages hors contexte). Les APNs, quant à eux, correspondent exactement aux langages hors contextes, ils sont donc très étudiés. Par défaut, un AP désigne un APN.

7.2 Automate à pile (non-déterministe)

La différence entre un AF classique et un AP se résume à deux changements : 1) il y a un alphabet supplémentaire pour désigner les symboles que l'on utilise dans la pile et 2) la fonction de transition devient plus complexe. Formellement, un **automate à pile non déterministe** est un 6-tuple $(Q, \Sigma, \Gamma, \delta, q_0, F)$ où

- Q est un ensemble fini d'états,
- Σ est l'alphabet d'entrée (habituel),
- Γ est l'alphabet de pile,
- $\delta : Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$ est la fonction de transition (discutée plus bas),
- q_0 est l'état initial,
- F est l'ensemble des états finaux avec $F \subseteq Q$,

Rappelons-nous que Σ_ε désigne $\Sigma \cup \{\varepsilon\}$. Pour les AFNs, cet ajout permet de définir des transitions où l'on ne lit aucun symbole (ε-transitions). De la même manière, on définit Γ_ε comme étant $\Gamma \cup \{\varepsilon\}$, ce qui permet de définir des transitions qui utilisent la pile et d'autres qui ne l'utilisent pas.

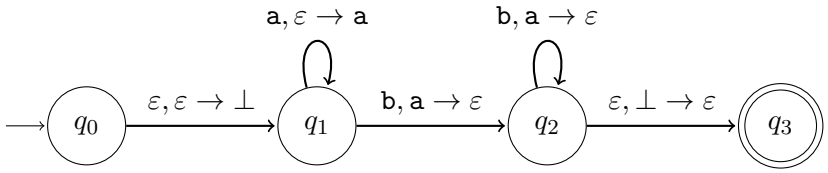
Examinons la signature de la fonction de transition δ, à savoir $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$. Le domaine est $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$. Concrètement, une transition dépend de 1) l'état courant, 2) le symbole lu sur l'entrée (ou rien, si ε), et 3) la *valeur lue au sommet de la pile* (ou rien, si ε). La lecture d'un symbole sur la pile a toujours pour effet de le dépiler (il est enlevé de la pile). Le co-domaine de δ est $\mathcal{P}(Q \times \Gamma_\varepsilon)$, ce qui signifie que plusieurs actions sont potentiellement possibles et seront toutes effectuées de manière non-déterministe. Pour chacune de ces actions, **l'automate tout entier (incluant la pile)** peut être vu comme **dupliqué** dans un univers parallèle. Chaque action, elle-même de la forme $Q \times \Gamma_\varepsilon$, revient à se déplacer sur un nouvel état et empiler au passage un symbole sur la pile (ou rien, si ε).

7.3 Exemples

7.3.1 AP reconnaissant le langage $\{a^n b^n \mid n \geq 1\}$

Intuitivement, on peut reconnaître ce langage comme suit : au départ, on empile un symbole $\perp$ (qui représentera le fond de la pile), puis, tant qu'on lit des `a`, on empile le symbole `a`. Puis, tant qu'on lit des `b`, on dépile les symboles `a` précédemment empilé. Si le sommet de la pile vaut $\perp$ quand le mot se termine, on accepte (on a lu autant de `b` que le nombre de `a` empilés).

Plus concrètement, on considère l'alphabet d'entrée $\Sigma = \{\texttt{a}, \texttt{b}\}$ et l'alphabet de pile $\Gamma = \{\texttt{a}, \perp\}$. Notre automate a 4 états $Q = \{q_0, q_1, q_2, q_3\}$, avec q_0 l'état initial et q_3 le seul état final. Les transitions sont comme illustrées sur l'automate suivant :

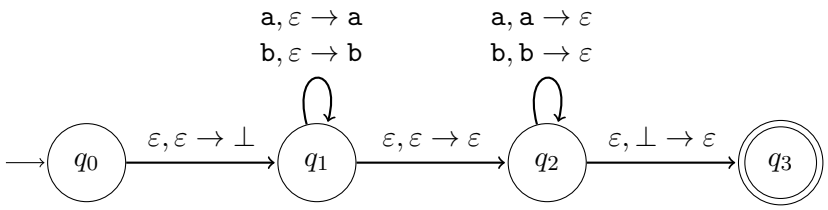


Chaque transition est représentée sous la forme $x, y \to z$, où x correspond au symbole lu sur le mot d'entrée (comme d'habitude), y correspond au symbole requis au sommet de la pile, qui sera dépilé (ou ε si l'on ne souhaite pas consulter la pile), et z correspond au symbole que l'on empilera au sommet de la pile si la transition est effectuée (ou ε si l'on ne souhaite rien ajouter à la pile). Ainsi, la description $\texttt{b}, \texttt{a} \to \varepsilon$ signifie "si on lit `b` sur le mot d'entrée et que le sommet de la pile vaut `a`, alors on dépile ce symbole `a` (c'est implicite) et on empile rien à la place".

Prenons chaque étape l'une après l'autre : depuis l'état q_0, sans ne rien lire sur le mot d'entrée ni sur la pile, on empile le symbole $\perp$ et on passe dans l'état q_1. Puis, sur l'état q_1, si on lit `a`, on ne dépile rien et on empile `a`. (Potentiellement, cela se répète plusieurs fois.) Toujours depuis q_1, si on lit un `b` sur le mot d'entrée ET que le sommet de la pile vaut `a`, alors ce `a` est dépilé, on empile rien à la place, et on passe dans l'état q_2. La même transition peut se répéter sur q_2. Enfin, si l'on rencontre le symbole $\perp$ sur le sommet de la pile, alors sans rien lire et sans rien empiler, on passe sur l'état q_3. On accepte alors si le mot est terminé. Comme pour les AFNs et les AFDs, les transitions non spécifiées reviennent à rejetter le mot directement, par exemple, si on lit un `a` sur l'entrée depuis l'état q_2, ou si on continue à lire des choses depuis q_3, le mot est rejeté.

7.3.2 AP reconnaissant les palindromes de longueur paire

L'automate suivant reconnaît le langage $L = \{w \cdot w^R \mid w \in \{\texttt{a}, \texttt{b}\}^*\}$, autrement dit les palindromes de longueur paire sur l'alphabet $\Sigma = \{\texttt{a}, \texttt{b}\}$:



Le principe est le suivant : sur la première moitié du mot, on empile les symboles lus. Puis, sur l'autre moitié, on vérifie en dépilant qu'on a bien les mêmes symboles (dans l'ordre inverse). Mais comment sait-on où se trouve la moitié ? C'est là que le non-déterminisme est essentiel. Grâce au non-déterminisme (incarné ici par la transition entre q_1 et q_2), l'automate se dédouble à chaque passage sur q_1, de sorte qu'au moins une exécution passe sur q_2 quand le vrai milieu est atteint et le mot sera accepté (s'il s'agit d'un palindrome). Rappelons qu'avec le non-déterminisme, il est suffisant qu'une des exécutions accepte le mot.

7.4 APD versus APN ?

Dans ce cours, nous nous sommes concentrés sur les automates à pile non-déterministes (APN), car ils correspondent aux grammaires hors contexte, tandis que les APD ne reconnaissent qu'un sous-ensemble des langages hors contexte. Par ailleurs, il est plus facile de voir les APDs comme des cas particuliers d'APNs.

La subtilité est que les APDs, bien que déterministes, utilisent quand même le symbole ε dans la définitions de leurs transitions (c'est une différence importante avec les AFDs). Pourquoi ? Parce-qu'on pourrait vouloir effectuer des traitements sur la pile (empiler ou dépiler) sans pour autant lire de symbole sur l'entrée, ou inversement, on pourrait vouloir lire un symbole sur l'entrée sans toucher à la pile, le tout sans jamais engendrer deux exécution distinctes. En effet, le point important est que l'automate ne doit avoir qu'une exécution possible, c'est tout ! Un APD est donc un cas particulier d'APN pour lequel un seul choix de transition existe depuis chaque état.

Notez que c'est le cas, par exemple, de l'AP reconnaissant le langage $a^n b^n$ plus haut : bien qu'il utilise des ε un peu partout, l'exécution ne prendra qu'un seul chemin. Cet AP est donc un APD. L'AP reconnaissant les palindromes, en revanche, est réellement non-déterministe, et nous avons besoin de cela pour deviner où se trouver le milieu du mot.

7.5 Annexe : utilisation d'une pile

Les piles sont des structures de données très utilisées en informatique. Elles se fondent sur le principe "dernier arrivé, premier sorti" (en anglais LIFO pour last in, first out), ce qui veut dire que le dernier élément ajouté à la pile est toujours le premier à en ressortir. Ce fonctionnement est moins puissant qu'une mémoire où l'on pourrait accéder à tout élément

de manière directe. Parfois, cette limitation est intrinsèquement voulue, comme ici, pour reconnaître les langages hors contexte, mais rien de plus (nous montrerons cela la semaine prochaine). Parfois, cela simplifie juste le traitement à effectuer.

Quelques exemples d'utilisation :

- Lorsqu'une fonction dans un programme appelle une autre fonction, qui appelle une autre fonction, cette exécution est gérée par le système d'exploitation à l'aide d'une pile (la pile d'exécution). À chacun de ces appels, on sauvegarde le contexte de la fonction appelante (les valeurs actuelles de ses variables, l'endroit où l'on se trouve de l'exécution, etc.) sur la pile, puis on commence l'exécution de la nouvelle fonction. Si cette dernière en appelle une autre, on empile à nouveau. Puis, lorsqu'une de ces fonctions termine, on dépile la précédente pour reprendre son exécution là où elle s'était arrêtée.
- On peut aussi utiliser une pile, par exemple, pour inverser facilement le contenu d'un tableau : parcourir tout le tableau en empilant ses éléments au fur et à mesure dans une pile, puis le parcourir à nouveau en y écrivant les éléments dépilées au fur et à mesure, ce qui aura pour effet d'inverser l'ordre des élements dans le tableau.
- Une pile d'assiette dans votre armoire fonctionne également comme cela.