Langages formels (11X003) - Automne 2026

# 7. Automates à pile

*Enseignant: Arnaud Casteigts*

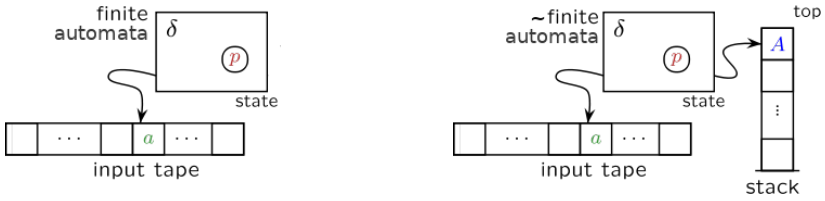
*Assistants: E. Arm & M. Marseloo*
*Moniteurs: A. Hugener & O. Oternaud*

(La partie sur les grammaires régulières a été ajoutée aux notes de la semaine dernière, Cours 6 sur les grammaires formelles).

Dans ce cours, nous revenons sur les machines. Nous avons vu que certains langages ne peuvent pas être reconnus par des automates finis. 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

Let automates que nous connaissons sont composés de deux parties : la partie contenant les états et les transitions et la partie sur laquelle on lit le mot en entrée, composée d'une bande (*tape*, en anglais) et d'une tête de lecture, ces deux éléments n'étant souvent pas dessinés (Figure 1a). La principale limitation de ces automates est qu'ils n'ont pas de mémoire. Les automates à pile (AP) consistent à augmenter ces machines avec une mémoire, manipulée comme une **pile** (ou *stack*, en anglais). À chaque transition, en plus de ses opérations habituelles, l'automate peut **empiler** ou **dépiler** un symbole au **sommet** de la pile. Et surtout, l'activation ou non d'une transition peut dépendre de la valeur qui s'y trouve (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). Cependant, les deux *ne sont pas équivalents* : ici, le non-déterminisme augmente réellement les capacités de la machine. On appelle langages hors

contexte *déterministes* les langages reconnaissables par des APDs. Les APNs, quant à eux, correspondent exactement aux langages hors contextes. Par défaut, un AP désigne un AP$\underline{\text{N}}$.

## 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,
- $\Sigma$ est l'alphabet d'entrée (habituel),
- $\Gamma$ 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 $\Sigma_\varepsilon$ désigne $\Sigma \cup \{\varepsilon\}$. Pour les AFNs, cet ajout permet de définir des transitions où l'on ne lit aucun symbole ($\varepsilon$-transitions). De la même manière, on définit $\Gamma_\varepsilon$ comme étant $\Gamma \cup \{\varepsilon\}$, ce qui permet d'autoriser des transitions qui n'utilisent pas la pile.

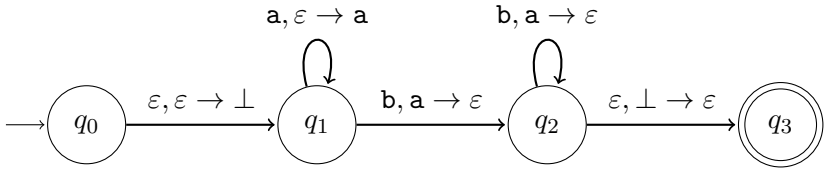
Examinons la signature de la fonction de transition $\delta$, à savoir $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \mathcal{P}(Q \times \Gamma_\varepsilon)$. Le domaine est $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon$. Concrètement, une transition dépend de 1) l'état courant, 2) le symbole lu sur l'entrée (ou rien, si $\varepsilon$), et 3) la *valeur lue au sommet de la pile* (ou rien, si $\varepsilon$). 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 $\delta$ 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, on parle alors de différentes **branches d'exécution**. 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 $\varepsilon$).

## 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 $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ et l'alphabet de pile $\Gamma = \{\mathtt{a}, \bot\}$. 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 :

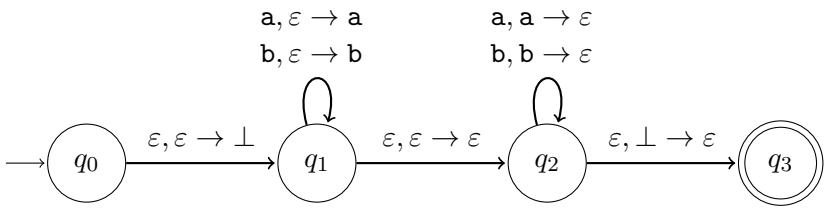



Chaque transition est représentée sous la forme $x, y \to z$, où $x$ correspond au symbole lu sur le mot d'entrée, $y$ correspond au symbole dépilé du sommet de la pile (rien si $\varepsilon$), et $z$ correspond au symbole que l'on empilera au sommet de la pile si la transition est effectuée (rien si $\varepsilon$). Ainsi, la description $\mathtt{b}, \mathtt{a} \to \varepsilon$ entre $q_1$ et $q_2$ signifie "si on est dans $q_1$, qu'on lit $\mathtt{b}$ sur le mot d'entrée et que le sommet de la pile vaut $\mathtt{a}$, alors on dépile ce symbole, on n'empile rien à la place et on se déplace sur $q_2$".

Reprenons les étapes l'une après l'autre : depuis l'état $q_0$, sans ne rien lire sur le mot d'entrée et sans rien dépiler, on empile le symbole $\bot$ et on passe dans l'état $q_1$. On peut voir cela comme une étape d'initialisation. Une fois sur l'état $q_1$, tant qu'on lit $\mathtt{a}$ sur le mot d'entrée, on empile $\mathtt{a}$ sur la pile en restant sur $q_1$ (boucle). Si on lit un $\mathtt{b}$ sur le mot d'entrée et que le sommet de la pile vaut $\mathtt{a}$, alors ce $\mathtt{a}$ est dépilé et on passe dans l'état $q_2$. La même opération se répètera sur $q_2$ tant que c'est possible (boucle). Si à un moment on rencontre le symbole $\bot$ sur le sommet de la pile, alors on passe sur l'état $q_3$. Si la lecture du mot est alors terminée, on accepte. Comme pour les AFNs, les transitions non spécifiées reviennent à faire rejetter la branche d'exécution actuelle. Par exemple, si on lit un $\mathtt{a}$ sur l'entrée depuis l'état $q_2$, ou si on continue à lire des choses depuis $q_3$, la branche rejette. Comme pour les AFNs, un mot est accepté si et seulement si au moins une branche d'exécution l'accepte.

### 7.3.2 AP reconnaissant les palindromes de longueur paire

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




Notez qu'on retrouve le même mécanisme d'initialisation que précédemment (utilisant le symbole $\bot$). Ce mécanisme n'est pas systématique, mais nous l'utiliserons souvent. Le

principe de cet automate 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). La difficulté est de deviner lorsqu'on a atteint la moitié du mot. C'est là que le non-déterminisme entre en jeu. À chaque fois qu'on lit un symbole depuis $q_1$, l'exécution se duplique en deux branches distinctes, l'une qui suppose Grâce au non-déterminisme (causé par la transition $\varepsilon, \varepsilon \to \varepsilon$ entre $q_1$ et $q_2$), l'automate se dédouble à chaque passage sur $q_1$, de sorte qu'au moins une exécution passera sur $q_2$ quand le vrai milieu est atteint et cette branche d'exécution là acceptera le mot s'il est un palindrome.

## 7.4 Clarification sur le non-déterminisme

Nous avons donné deux exemples d'automates à pile ci-dessus. Le premier est en fait déterministe, tandis que le second ne l'est pas. Le fait que le premier est bien un APD peut sembler perturbant, car il utilise $\varepsilon$ dans ses transitions. Le point important à réaliser ici est que le fait d'être déterministe ou non-détermiste est lié au fait d'avoir plusieurs choix. Dans le premier automate, aucun exécution possible d'aboutira à créer plusieurs branches, il est donc déterministe. En revanche, le second automate n'est pas déterministe, car la transition du milieu $\varepsilon, \varepsilon \to \varepsilon$ créée réellement plusieurs branches d'exécution (l'une qui reste sur $q_1$, l'autre qui passe sur $q_2$).

Comme évoqué plus haut, la puissance des APDs est strictement inférieure à celle des APNs. Cependant, nous ne ne discuterons pas ce point en détail, en nous concentrant sur les APNs en général, car ils correspondent exactement aux langages que l'on peut exprimer avec des grammaires hors-contexte.

## 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.