Langages formels (11X003) - Automne 2026

8. Équivalence AP - GHC

Enseignant: Arnaud Casteigts

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

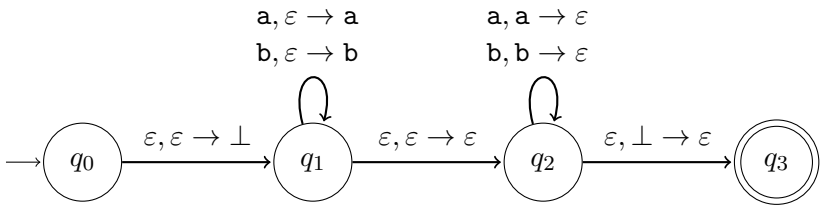
Pour rappel, nous désignons par AP un automate à pile non-déterministe (APN). Dans ce cours, nous allons montrer que n'importe quel langage engendré par une grammaires hors-contexte (GHC) peut être reconnu par un AP. Une transformation des AP vers les GHC existe aussi, ce qui établit que les GHC et les AP sont bien capables de décrire les mêmes langages. Nous ne verrons pas cette seconde transformation, qui est compliquée.

8.1 Automates à pile (complément)

Nous commençons par une illustration complète du fonctionnement d'un APN. Puis nous mentionnons un modèle légèrement plus pratique (mais équivalent) qui permet d'écrire plusieurs symboles sur la pile d'un seul coup.

8.1.1 Illustration d'une exécution

Voici l'AP de la semaine dernière, qui reconnaît les palindromes de longueur paire sur l'alphabet $\Sigma = \{a, b\}$, autrement dit le langage $L = \{ww^R \mid w \in \Sigma^*\}$. Cet automate possède notamment une transition non-déterministe entre q_1 et q_2 qui lui sert à "deviner" où se trouve le milieu du mot d'entrée.

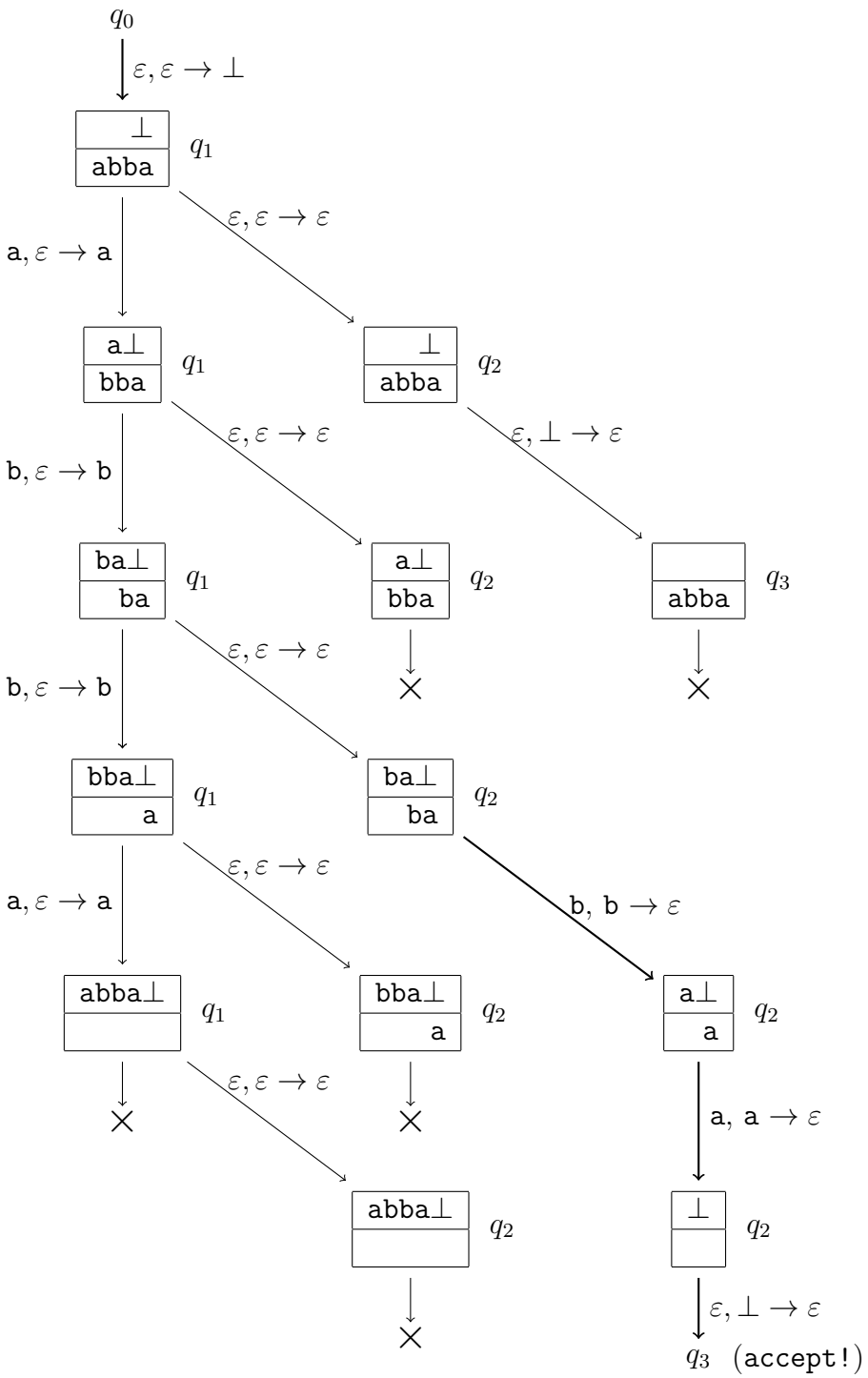


Observons l'exécution de cet automate lorsqu'on lit le mot `abba`. Pour simplifier, nous représentons la configuration de l'automate à un instant donné par deux éléments :

contenu de la pile
mot restant à lire

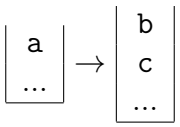
Pour le "contenu de la pile", le symbole le plus à gauche correspond au sommet de la pile. Le "mot restant à lire" correspond aux symboles qui n'ont pas encore été lus sur le mot d'entrée.

Initialement, le contenu de la pile est vide et le mot restant à lire est le mot d'entrée lui-même, ici `abba`. L'exécution étant non-déterministe, on peut la représenter par un arbre, où chaque branchement sur plusieurs successeurs correspond à un choix parmi plusieurs transitions possibles.



8.1.2 Empilement de plusieurs symboles d'un coup

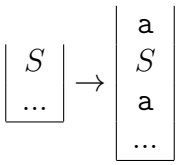
Dans certains cas, il est plus pratique d'autoriser des transitions qui empilent *plusieurs* symboles à la fois. C'est juste par facilité, car on peut toujours convertir un tel automate en un AP normal qui n'empile qu'un symbole à la fois (en ajoutant des états intermédiaires). Concrètement, cela permet des transitions du type $x, y \to z$, où x est un symbole d'entrée (ou ε) ; y est un symbole de pile (ou ε) ; et z est un *mot* sur l'alphabet de pile. Ce mot est empilé de la dernière à la première lettre. Par exemple, "$\varepsilon, \texttt{a} \to \texttt{bc}$" transforme la pile comme suit (mentalement, `a` est remplacé par `bc`) :



8.2 Grammaire hors-contexte → Automate à pile

Soit une grammaire hors-contexte $G = (V, \Sigma, S, \mathcal{P})$, nous allons transformer G en un automate à pile $A = (Q, \Sigma, \Gamma, \delta, q_0, F)$ (qui peut empiler plusieurs symboles d'un coup) tel que A reconnaît un mot si et seulement si ce mot peut être engendré par G.

Intuitivement, l'automate simule les dérivations possibles de la grammaire, le mot intermédiaire étant stocké dans la pile. Lorsqu'une variable se trouve au sommet de la pile, une transition la remplace selon l'une des règles de G. Par exemple, si la grammaire possède une règle $S \to aSa$ et si S est au sommet de la pile, S peut être remplacé par aSa comme suit (indépendamment du reste de la pile) :



Si plusieurs règles sont applicables, on les considère *toutes* de manière non-déterministe, chaque transition engendrant un branchement distinct dans un nouvel univers. Ainsi, pour chaque *dérivation* possible de la grammaire, il existe une exécution correspondante.

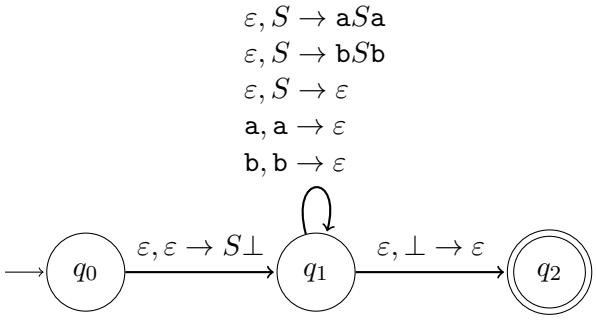
La difficulté est de savoir quoi faire des symboles terminaux qui se trouvent au sommet de la pile, qui nous empêchent d'accéder aux variables plus bas. L'astuce ici est de réaliser que ces symboles ne changeront plus et il correspondent exactement aux symboles qu'il faut lire sur l'entrée à ce stade du mot. On utilise donc d'autres transitions qui "consomment" ces symboles en avançant la lecture du mot, et ce, jusqu'à ce qu'une variable remonte au sommet. Ainsi, on alterne les transitions qui simulent la grammaire et celles qui lisent le mot d'entrée. Si le mot d'entrée se termine en même temps que la pile est vide, c'est qu'il existe

une dérivation de la grammaire qui produit ce mot. L'automate accepte donc ce mot. Si à tout moment les symboles terminaux ne correspondent pas au mot d'entrée, l'exécution de cette branche non-déterministe est détruite. Le mot d'entrée est considéré comme accepté si au moins une branche de l'exécution l'accepte (comme pour les AFNs).

Prenons à nouveau l'exemple des palindromes de longueur paire sur l'alphabet $\Sigma = \{a, b\}$, en supposant qu'on ne connaît pas l'automate correspondant, mais seulement la grammaire qui l'engendre. En l'occurence, nous partons donc de la grammaire $S \to aSa \mid bSb \mid \varepsilon$. La technique est la même pour toutes les grammaires : on construit un automate à pile qui a trois états q_0, q_1, q_2. La transition de q_0 à q_1 commence par ajouter un symbole $\perp$ au fond de la pile et à empiler le symbole de départ S par dessus. Ensuite, on ajoute un certain nombre de transitions qui bouclent sur q_1, à savoir :

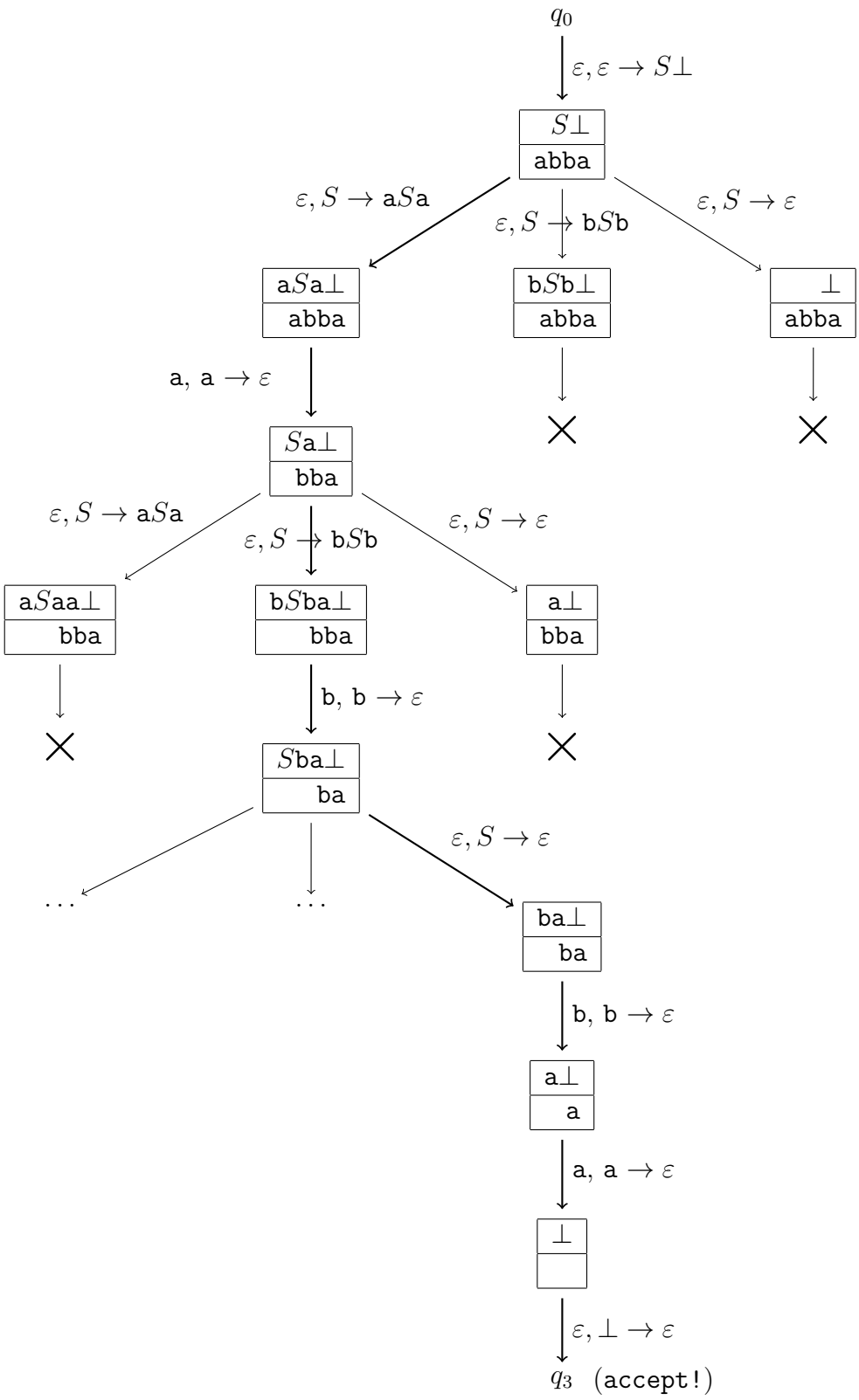
- Une transition "$\varepsilon, \alpha \to \beta$" pour chaque règle $\alpha \to \beta$ de la grammaire.
- Une transition "$\mathtt{s}, \mathtt{s} \to \varepsilon$" pour chaque symbole $\mathtt{s} \in \Sigma$.

Puis on ajoute une transitions de q_1 vers q_2 qui atteint l'état final si la pile est vide. Dans notre exemple, cela donne l'automate suivant :



Les trois premières transitions qui bouclent sur q_1 simulent la grammaire lorsqu'une variable est au sommet de la pile. Les deux autres font avancer la lecture du mot d'entrée lorsque des symboles terminaux sont au sommet (à condition que les deux soient identiques). Cet automate acceptera un mot w si et seulement si $S \overset{*}{\Rightarrow} w$ (c.à.d. w peut être dérivé depuis S), ce qui correspond bien à l'objectif.

Observons maintenant l'exécution de cet automate sur un mot d'entrée, en prenant à nouveau l'exemple du mot `abba`.


q_0
$\varepsilon, \varepsilon \to S\bot$
$S\bot$ abba
$\varepsilon, S \to \mathsf{a}S\mathsf{a}$
$\varepsilon, S \to \mathsf{b}S\mathsf{b}$
$\varepsilon, S \to \varepsilon$
$\mathsf{a}S\mathsf{a}\bot$ abba
$\mathsf{b}S\mathsf{b}\bot$ abba
$\bot$ abba
$\mathsf{a}, \mathsf{a} \to \varepsilon$
$S\mathsf{a}\bot$ bba
$\varepsilon, S \to \mathsf{a}S\mathsf{a}$
$\varepsilon, S \to \mathsf{b}S\mathsf{b}$
$\varepsilon, S \to \varepsilon$
$\mathsf{a}S\mathsf{aa}\bot$ bba
$\mathsf{b}S\mathsf{ba}\bot$ bba
$\mathsf{a}\bot$ bba
$\mathsf{b}, \mathsf{b} \to \varepsilon$
$S\mathsf{ba}\bot$ ba
$\varepsilon, S \to \varepsilon$
...
...
$\mathsf{ba}\bot$ ba
$\mathsf{b}, \mathsf{b} \to \varepsilon$
$\mathsf{a}\bot$ a
$\mathsf{a}, \mathsf{a} \to \varepsilon$
$\bot$
$\varepsilon, \bot \to \varepsilon$
q_3 (accept!)

8.3 Automate à pile $\rightarrow$ Grammaire hors-contexte

Cette transformation n'est pas au programme, vous n'êtes donc pas tenus de la connaître, cependant vous devez savoir qu'elle existe !

8.4 Remarques générales sur les langages hors-contextes

Si L est un langage hors-contexte (h.c.), alors :

- L^* est h.c.
- $\overline{L}$ n'est pas forcément h.c.

Si L_1 et L_2 sont des langages hors-contextes, alors :

- $L_1 \cup L_2$ est h.c.
- $L_1 \circ L_2$ est h.c.
- $L_1 \cap L_2$ n'est pas forcément h.c.

Cependant, si L_1 est *régulier* et L_2 est h.c., alors $L_1 \cap L2$ est h.c.

Les opérations sur les langages hors-contextes sont donc à manipuler avec précaution, car elles ne sont pas toutes closes par les opérations usuelles. Notamment, l'intersection et la complémentation peuvent produire des langages qui ne sont pas hors-contexte.