# Automates à pile,
# analyse syntaxique universelle

Licence info S5

UFR IEEA TD COMPIL – 2009-2010

## Exercice 1 : Automates à pile généraux

Donner un automate à pile (acceptant par pile vide ou état final) pour les langages suivants, quand cela est possible :
- $\{a^n c^m b^n \mid n > 0, m \geq 0\}$ ;
- $\{wcw \mid w \in \{a, b\}^*\}$.

## Exercice 2 : Automate des items

On considère la grammaire $G$ d'axiome $S$, de terminaux $\{a, b, d, e\}$ et de productions :
$S \to AB \mid Da \quad A \to aAb \mid \epsilon \quad B \to bB \mid \epsilon \quad D \to dD \mid e$

**Q 2.1** : Donner un arbre syntaxique pour le mot $ab$. □

**Q 2.2** : Donner l'ensemble des items $It_G$ en précisant l'item initial et l'item final □ .

**Q 2.3** : Quels sont les trois types d'opérations réalisées par l'automate des items ? □

**Q 2.4** : Donner la suite des piles résultant de l'analyse réussie du mot $ab$ par l'automate des items associé à $G$. □

## Exercice 3 : Un analyseur universel : l'algorithme CYK

L'algorithme CYK a été proposé par Cocke, Younger et Kasami en 1965. Il réalise une analyse syntaxique « universelle » : pour un mot $w$ et une grammaire algébrique $G$, il répond à la question $w \in L(G)$, sans prérequis pour $G$ autre que sur sa forme (contrairement aux analyseurs ascendants et descendants). L'algorithme fonctionne en effet pour toute grammaire en *forme normale de Chomsky* (FNC, CNF en anglais), et toute grammaire algébrique peut être mise sous cette forme.

### 3.1 : Forme Normale de Chomsky (FNC)

Une grammaire est en FNC si $P \subseteq V_N \times ({V_N}^2 \cup V_T)$, c'est à dire si ses productions sont de la forme :
- $X \to a$ avec $a \in V_T$ ;
- $X \to X_1X_2$ avec $\{X_1, X_2\} \subseteq V_N$.

Pour mettre une grammaire en FNC, on part d'une grammaire réduite et *propre*. Une grammaire est propre si elle ne contient pas de production de la forme :
- $X \to Y$ avec $Y \in V_N$ ;
- $X \to \epsilon$.

Il existe un algorithme pour obtenir à partir d'une grammaire algébrique une grammaire propre équivalente (éventuellement à $\epsilon$ près) ; on ne le verra pas ici.

**Mise sous forme de Chomsky** On procède en 2 étapes.

1. On introduit pour chaque terminal $a$ un nouveau non-terminal $X_a$ qui le représente, en remplaçant partout dans les productions $a$ par $X_a$ puis en ajoutant la production $X_a \to a$. Si $a$ est déjà représenté par un non-terminal (production $X \to a$) alors il est inutile d'introduire un nouvel $X_a$, on utilise $X$. À l'issue de cette étape, puisqu'on part d'une grammaire propre, les productions sont de la forme :
   - $X \to a$ avec $a \in V_T$ ;
   - $X \to X_1 \ldots X_n$ avec $\{X_1, \ldots X_n\} \subseteq V_N$ et $n \geq 2$.
2. On découpe le membre droit des productions de la forme $X_1 \ldots X_n$ avec $n > 2$ pour obtenir un membre droit de la forme $X_1X_2$. Pour cela, on introduit un nouveau terminal $Z$, on remplace $X_1 \ldots X_n$ par $X_1Z$, on ajoute la production $Z \to X_2 \ldots X_n$, et on recommence tant que nécessaire. Par exemple on remplacera $A \to BCDE$ par $\{A \to BF, F \to CDE\}$, puis par $\{A \to BF, F \to CG, G \to DE\}$.

**Q 3.1** : Mettre la grammaire $G_E$ donnée par les règles suivantes en forme normale de Chomsky :

$$E \to E{+}E \mid E{*}E \mid (E) \mid \mathtt{i}$$

□

## 3.2 : L'algorithme CYK

Cet algorithme ascendant (il part du mot à reconnaître et remonte vers l'axiome) opère sur une grammaire algébrique en FNC. L'intuition est la suivante. Soit la grammaire d'axiome $S$ et de productions :

$$S \to c \,|\, d \,|\, AB \,|\, BB \qquad A \to a \,|\, AA \qquad B \to b \,|\, BB$$

On veut savoir si un mot $w$ est engendré par l'axiome. Si $w$ est de longueur 1, alors les productions à utiliser sont $S \to c$ ou $S \to d$, et le mot est soit $c$ soit $d$. Si $w$ est de longueur $n$ avec $n$ valant au moins 2, alors soit $AB \Rightarrow^* w$ (production $S \to AB$), soit $BB \Rightarrow^* w$ (production $S \to BB$). Il existe donc un découpage du mot en deux sous-mots $w_1$ et $w_2$ différents de $\epsilon$ tels que :

- soit $A \Rightarrow^* w_1$ et $B \Rightarrow^* w_2$ ;
- soit $B \Rightarrow^* w_1$ et $B \Rightarrow^* w_2$.

Il existe $n-1$ découpages possibles de $w$ en $w_1$ et $w_2$ : il faut tous les essayer, et opérer récursivement pour $w_1$ et $w_2$, jusqu'à tomber sur des sous-mots de longueur 1.

On suppose que $w$ est de la forme $a_1 a_2 \ldots a_n$. L'algorithme fonctionnant de manière ascendante, il va commencer par identifier quels fragments de longueur 1 (les $a_i$) se dérivent de quels non-terminaux. Ensuite, si on sait que $Y \Rightarrow^* a_i \ldots a_j$, que $Z \Rightarrow^* a_{j+1} \ldots a_k$ et que $X \to YZ$ est une production de la grammaire alors on peut en déduire que $X \Rightarrow^* a_i \ldots a_k$. L'algorithme applique ce principe itérativement pour toutes les longueurs $l$ possibles des fragments du mot à analyser $a_1 a_2 \ldots a_n$, c'est-à-dire pour $l$ allant de 1 à $n$.

L'entité de base de l'algorithme est la *cellule*. Une cellule est identifiée par un couple d'entiers $(l, i)$ avec $l, i \leq n$ ($l$ pour *longueur* et $i$ pour *indice*). Une cellule contient un ensemble de non-terminaux. La cellule $(l, i)$ contient le non-terminal $X$ ssi $X$ peut engendrer le fragment du mot $a_1 a_2 \ldots a_n$ de longueur $l$ et commençant à l'indice $i$ :

$$X \in (l, i) \text{ ssi } X \Rightarrow^* a_i \ldots a_{i+l-1}$$

Le mot $a_1 a_2 \ldots a_n$ est engendré par la grammaire ssi, à l'issue de l'algorithme, l'axiome appartient à la cellule $(n, 1)$.

À la première itération $l = 1$, les fragments sont les terminaux. On ajoute $X$ à la cellule $(1, i)$ si $X \to a_i$ est une production de la grammaire. Pour les itérations suivantes (fragments de longueur $l > 1$), pour chaque $i$, on ajoute $X$ à la cellule $(l, i)$ ssi il existe un découpage de $a_i \ldots a_{i+l-1}$ en $a_i \ldots a_{i+m-1}$ et $a_{i+m} \ldots a_{i+l-1}$ (pour $m$ compris entre 1 et $l-1$) tel que :

- $Y \Rightarrow^* a_i \ldots a_{i+m-1}$ : la cellule $(m, i)$ contient $Y$ ;
- $Z \Rightarrow^* a_{i+m} \ldots a_{i+l-1}$ : la cellule $(l-m, i+m)$ contient $Z$ ;
- $X \to YZ$ est une production de la grammaire.

Au final l'algorithme est le suivant :

```
analyseCYK(a_1 ... a_n : un mot , G = (V_T, V_N, S, P) : une grammaire en FNC) : booléen
    pour i allant de 1 à n faire
        si X → a_i ∈ P alors mettre X dans la cellule (1, i) ; fin si ;
    fin pour ;
    pour l allant de 2 à n faire
        pour i allant de 1 à n − l + 1 faire
            pour m allant de 1 à l − 1 faire
                si Y dans (m, i) et Z dans (l − m, i + m) et X → YZ ∈ P alors
                    ajouter X à la cellule (l, i) ;
                fin si ;
            fin pour ;
        fin pour ;
    fin pour ;
    retourner (la cellule (n, 1) contient S) ;
fin analyseCYK ;
```

**Q 3.2** : En stockant les cellules dans une matrice carrée de taille $n$, dérouler l'algorithme CYK sur la grammaire $G_E$ et le mot $w_E$ `i+i*i`. □

**Q 3.3** : $w_E$ est ambigu. Quelle interprétation l'algorithme a-t-il « choisi » ? □

**Q 3.4** : Si on considère la taille de la grammaire comme constante, donner en fonction de la longueur $n$ du mot à analyser la complexité de la fonction `analyseCYK`. □