Dans ce qui suit, nous allons introduire les automates à pile ainsi que les grammaires algébriques appelées aussi grammaires à contexte libre ou grammaires de type 2.

1. Automate à pile

1.1 Définition

Un automate à pile est composé d'un organe de commande à états finis, d'une bande en entrée qui contient les mots à reconnaître et une mémoire de type pile (Figure 1).

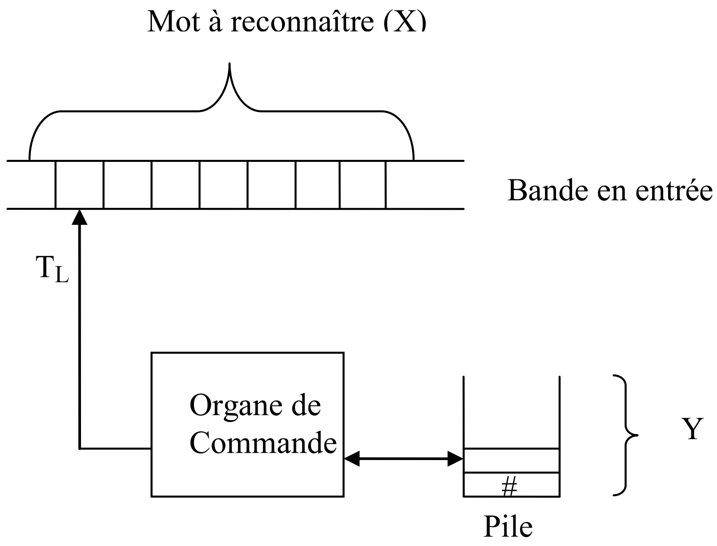

Figure 1. Automate à pile

Plus formellement, un automate à pile est un 7-uplets <X, Y, S, S_0, F, II, #> où :

X est l'alphabet d'entrée,
Y est l'alphabet auxiliaire (alphabet de la pile),
S est l'ensemble des états de l'automate,
S_0 est l'état initial, $S_0 \in S$
F est l'ensemble des états finaux, $F \subseteq S$
II est l'ensemble des instructions, $II : S \times (X \cup \{\varepsilon\}) \times Y \rightarrow S \times Y$
est le symbole de pile vide.

1.2 Opérations sur la pile

Trois opérations, définies ci-dessous, sont associées à l'automate à pile:

Lecture et empilement : $y_i\ S_i\ w_i \rightarrow y_i\ f(w_i)\ S_j$, $\quad yi \in Y, w_i \in X, S_i, S_j \in S$
Si le sommet de pile est y_i, si l'automate est à l'état S_i, et si la lettre w_i est sous la tête de lecture alors, l'automate empile w_i et passe à l'état S_j.

Lecture et dépilement : $y_i\ S_i\ w_i \rightarrow S_j$, $\quad yi \in Y, w_i \in X, S_i, S_j \in S$
Si le sommet de pile est y_i, si l'automate est à l'état S_i, et si la lettre w_i est sous la tête de lecture alors, l'automate dépile le sommet de pile y_i et passe à l'état S_j.

Lecture seulement : $y_i\ S_i\ w_i \rightarrow y_i\ S_j$, $\quad yi \in Y, w_i \in X, S_i, S_j \in S$

Si le sommet de pile est y_i, si l'automate est à l'état S_i, et la lettre w_i est sous la tête de lecture alors, l'automate passe à l'état S_j sans modifier la pile.

Exemple 1 :

Soit *Ap* <X, Y, S, S_0, F, II, #> un automate à pile où X = {a, b}, Y= {a, b, #}, S= {S_0 , S_1, S_f} F = { S_f}, et

P = { $\#\ S_0\ a \rightarrow \#\ a\ S_0$	*Empiler le premier a rencontré*
$a\ S_0\ a \rightarrow a\ a\ S_0$	*Tant qu'il y a des a en entrée, les empiler*
$a\ S_0\ b \rightarrow S_1$	*Dépiler le a à la rencontre d'un b et changer d'état*
$a\ S_1\ b \rightarrow S_1$	*Tant qu'il y a des b en entrée, dépiler les a*
$\#\ S_1 \rightarrow \#\ S_f$ }	*Plus de lettres en entrée et pile vide, aller à l'état final.*

Exemple 2 :

Soit *Ap* <X, Y, S, S_0, F, II, #> un automate à pile où X = {a, b}, Y = {a, b, #}, S = {S_0,S_1,S_f}, F = { S_f}, et

P = { $\#\ S_0\ a \rightarrow \#\ a\ S_0$	*Empiler le premier a rencontré*
$a\ S_0\ a \rightarrow a\ a\ S_0$	*Tant qu'il y a des a en entrée, les empiler*
$a\ S_0\ b \rightarrow S_1$	*Dépiler le a à la rencontre d'un b et changer d'état*
$a\ S_1\ b \rightarrow S_1$	*Tant qu'il y a des b en entrée, dépiler les a*
$a\ S_1 \rightarrow a\ S_f$ }	*Plus de lettres en entrée et au moins un a dans la pile, aller à l'état final*

1.3 Configuration d'un automate à pile

- Une configuration d'un automate à pile est un triplet (y, S_i, w), où y est le mot dans la pile, S_i est l'état courant dans lequel est l'automate et w est le mot qu'il reste à lire en entrée, $w \in X^*$ et $y \in Y^*$. La tête de lecture se trouve sous la première lettre de w.
- On appelle configuration initiale le triplet (#,S_0, w), la pile est vide, l'automate est à l'état initial S_0 et le mot à reconnaître en entrée est w, $w \in X^*$.
- On appelle configuration finale le triplet (y,S_f, ε), où y est le mot dans la pile $y \in Y^*$, S_f est un état final $S_f \in F$.

1.4 Lecture d'une lettre de X / Lecture d'un mot de X^*

Soit *Ap* <X, Y, S, S_0, F, II, #> un automate à pile. La lecture d'une lettre, $w_i \in X$, fait passer l'automate d'une configuration (yy_k, S_i, w_iw) à une configuration (yy_m, S_j, w), représentée comme suit : $yy_k\ S_i\ w_iw \underset{A_p}{\vdash} yy_m\ S_j\ w$ Si $(y_k\ S_i\ w_i \rightarrow y_m\ S_j) \in$ II., $y_k \in Y$

Nous définissons de plus la relation $\overset{*}{\underset{A_p}{\vdash}}$ pour la lecture d'un mot w de longueur n.

Prenons l'automate de l'exemple 1. Nous voulons exécuter l'automate sur le mot w = aabb. L'automate démarre d'une configuration initiale : L'automate est à l'état S_0, la pile est vide, et la tête de lecture est sous la première lettre du mot à reconnaitre (a). La première instruction qui consiste à empiler le a, et rester à l'état S0 est exécutée. La tête de lecture se déplace automatiquement vers la prochaine lettre à lire (deuxième a). Après exécution de l'instruction, l'automate passe à une nouvelle configuration # a S_0 abb. Le passage de l'automate de la configuration initiale à cette nouvelle configuration est représenté comme suit :

$$\# \; S_0 \; aabb \vdash_{A_p} \# \; a \; S_0 \; abb$$

Dans cette configuration, il y a un a dans la pile, l'automate est à l'état S_0 et la tête de lecture est sous le $2^{ème}$ a. On peut donc exécuter la $2^{ème}$ instruction et l'automate passe à une $3^{ème}$ configuration # aa S_0 bb.

$\# \; a \; S_0 \; abb \vdash_{A_p} \# \; aa \; S_0 \; bb$ Passage de l'automate de la 2ème configuration à la 3éme.

Le passage d'une configuration à l'autre se fait après exécution d'une instruction de l'ensemble II de l'Automate

1.5 Mot reconnu par un automate à pile

Soit *Ap* <X, Y, S, S_0, F, II, #> un automate à pile. Un mot w $\in X^*$ est reconnu par *Ap* si ce mot fait passer l'automate à pile de sa configuration initiale à une configuration finale :

$$\#, S_0, w \overset{*}{\vdash}_{A_p} y, S_f, \varepsilon$$

Exemple 3 :

Soit l'automate défini dans l'exemple 1. Le mot w = aabb est-il reconnu par l'automate ?

$$\underbrace{\# \; S_0 \; aabb}_{\text{Config. Initiale}} \vdash_{A_p} \# \; a \; S_0 \; abb \vdash_{A_p} \# \; aa \; S_0 \; bb \vdash_{A_p} \# \; a \; S_1 \; b \vdash_{A_p} \# \; S_1 \vdash_{A_p} \underbrace{\# \; S_f}_{\text{Config. Finale}}$$

Le mot aabb est reconnu par l'automate *Ap*.

Exemple 4 :

Le mot w = aab est-il reconnu par *Ap* ?

$$\underbrace{\# \; S_0 \; aab}_{\text{Config. Initiale}} \vdash_{A_p} \# \; a \; S_0 \; ab \vdash_{A_p} \# \; aa \; S_0 \; b \vdash_{A_p} \# \; a \; S_1$$

Après la lecture du b, on reste bloqué à l'état S_1. S1 n'est pas un état final et le mot aab n'est pas reconnu par *Ap*.

1.6 Langage reconnu par un automate à pile

Un langage reconnu par un automate à pile est défini comme suit :

$$L(Ap) = \{w \in X^* \text{ tq } (\#, S_0, w) \vdash^{*}_{A_p} (y, S_f, \varepsilon) \}$$

Exemple 5 : Le langage reconnu par l'automate défini par les instructions de l'automate de l'exemple 1 est le suivant : $L(G) = \{a^n b^n, n > 0\}$.

Exemple 6 : Le langage reconnu par l'automate défini par les instructions de l'automate de l'exemple 1 est le suivant : $L(G) = \{a^n b^m, n > m\}$.

1.7 Automate à pile vide

Définition :

On appelle automate à pile vide l'automate reconnaissant le langage suivant :

$$L(A_\varepsilon) = \{w \in X^* \text{ tq } (\#, S_0, w) \vdash^{*}_{A_p} (\#, S_f, \varepsilon) \}$$

Exemple 7 : L'automate de l'exemple 1 est un automate à pile vide

Théorème :

A tout automate à pile A_p <X, Y, S, S_0, F, II, #>, il existe une automate à pile vide $A_{p\varepsilon}$ <X, Y', S', S'_0, F', II', #> équivalent.

Démonstration :

1. Définissons les paramètres de l'automate à pile $A_{p\varepsilon}$:

$Y' = Y$

$S' = S \cup \{S_t\}, S_t \notin S$

$F' = \{S_t\}$

$II' = II \cup \{ y_i S_k \rightarrow S_t, \forall y_i \in Y, \forall S_k \in F\} \cup \{ y_i S_t \rightarrow S_t, \forall y_i \in Y-\{\#\}\}$

2. Montrons que $L(A_p) = L(A_{p\varepsilon})$

$\forall w \in L(A_p) \Leftrightarrow w \in L(A_{p\varepsilon})$

$w \in L(A_p) \Leftrightarrow \# S_0 w \vdash^{*}_{A_p} \# y_1 y_2 \dots y_n S_f$ avec $S_f \in F \Leftrightarrow \# S_0 w \vdash^{*}_{A_{p\varepsilon}} \# y_1 y_2 \dots y_n S_f \Leftrightarrow \# y_1 y_2 \dots y_n S_f \vdash_{A_{p\varepsilon}} \# y_1 y_2 \dots y_{n-1} S_t \vdash^{*}_{A_{p\varepsilon}} \# S_t \Leftrightarrow w \in L(A_{p\varepsilon})$

2. Grammaire algébrique

Une grammaire algébrique est un quadruplet G <X,V, P, S> où :

- X est l'alphabet,
- V un ensemble de variables,
- $S \in V$ l'axiome,
- P l'ensemble des règles de production de la forme :

$$A \rightarrow \alpha \text{ avec } A \in V \text{ et } \alpha \in (X \cup V)^*$$

Exemple 1: Soit G la grammaire algébrique où X = {a, b}, V={S, A}, et P est défini comme suit :
$\{S \rightarrow abSA \,/\, \varepsilon,\ A \rightarrow Aa \,/\, \varepsilon\}$

2.1 Dérivation gauche (respectivement dérivation droite)

Soit G<X,V,P,S> une grammaire algébrique. On dit qu'un mot w s'obtient par dérivation gauche (resp. dérivation droite) s'il est généré à partir de l'axiome en dérivant toujours le non terminal le plus à gauche (resp. le non terminal le plus à droite).

2.2 Arbre de dérivation

Soit G<X, V, P, S> une grammaire à contexte libre. Ar est un arbre de dérivation pour G si et seulement si :

- Chaque nœud de Ar est étiqueté d'un symbole appartenant à $X \cup V \cup \{\varepsilon\}$
- La racine de l'arbre est étiquetée S (axiome)
- Si un nœud interne est étiqueté A alors $A \in V$
- Si un nœud n est étiqueté A et ses fils $n_1, n_2, \ldots, n_n$ sont étiquetés respectivement $X_1, X_2, \ldots X_n$, alors $A \rightarrow X_1 X_2 \ldots X_n$ est une production de P
- Si un nœud est étiqueté ε, alors c'est le seul fils de son père.

2.3 Langage engendré par une grammaire algébrique

Le langage engendré par la grammaire algébrique G <X,V,P,S> est

$$L = L(G)= \{w \in X^* \text{ tq } S \vdash^{*}_{G} w \}.$$

Exemple 2: Le langage engendré par la grammaire du PASCAL est l'ensemble de tous les programmes qui sont correctement écrits en code source PASCAL.

2.4 Ambiguïté

Soit G<X,V,P,S> une grammaire algébrique :

Définition 1: un mot w est ambigu s'il existe, pour ce mot, plus d'une dérivation gauche (resp. droite) à partir de l'axiome.

Définition 2 : Une grammaire algébrique G<X,V,P,S> est ambiguë s'il existe un $w \in L(G)$ tel que w est ambigu.

Définition 3 : Un langage algébrique a une ambiguïté inhérente si toutes les grammaires qui l'engendrent sont ambiguës.

2.5 Simplification des grammaires algébriques

Soit G<X,V,P,S> une grammaire algébrique :

Définition 1: un symbole $Z \in (X \cup V)$ est inutile dans la grammaire G s'il n'existe aucune dérivation de la forme $S \overset{*}{\underset{G}{\vdash}} w_1 Z w_2 \overset{*}{\underset{G}{\vdash}} w_1 x w_2$, avec w_1, x, et $w_2 \in X^*$

Exercice 1: Donner un algorithme qui permet de définir toutes les variables de V inutiles.

Définition 2 : Une variable de V ou une lettre de X est dite inaccessible si elle n'apparaît dans aucune dérivation de G.

Exercice 2: Donner un algorithme qui permet de trouver toutes les variables inaccessibles de V.

Définition 3 : une variable A est dite non productive si elle ne génère aucun mot de X*.

Définition 4 : une grammaire sans variables inaccessibles et sans variables non productives est dite grammaire réduite.

Exercice 2: Donner un algorithme qui permet de construire une grammaire réduite G' à partir d'une grammaire G.

2.6 Grammaire ε-libre

Définition 1: une grammaire algébrique G <X, V P, S> est ε-libre si

1. P ne contient aucune production de type $A \to \varepsilon \ \forall A \in V$, ou
2. $S \to \varepsilon \in P$ et S n'apparaît dans aucun membre droit de production.

Proposition 1 : Soit G<X, V, P, S>, une grammaire algébrique non ε-libre, il existe une grammaire G'<X', V', P', S'> ε-libre équivalente.

Passage d'une grammaire algébrique non ε-libre vers une grammaire ε-libre.

Algorithme ε-libre (in G<X,V,P,S>, out G'<X,V',P', S'>)

Début

$P' = \varnothing$

1. Construire $V_\varepsilon = \{A \in V \ / \ A \overset{+}{\underset{G}{\vdash}} \varepsilon\}$

2. Construire P' de la manière suivante :

- **Si** $A \to \alpha_1 B_1 \alpha_2 B_2 \ldots \alpha_k B_k$ **∈ P**, $K \geq 0$, $\alpha_j \in (X \cup V)^*$ et ne contient aucun élément de V_ε et $B_j \in V_\varepsilon$ **alors** rajouter à P' toutes les productions de la forme

  $A \to \alpha_1 X_1 \alpha_2 X_2 \ldots \alpha_k X_k$ où $X_i = B_i$ ou $X_i = \varepsilon$

  sans rajouter la production $A \to \varepsilon$.

- **Si** $S \in V_\varepsilon$ **alors** rajouter à P' la production $S' \to S \ / \ \varepsilon$. S' devint le nouvel axiome de la grammaire. $V' = V \cup \{S'\}$

Sinon $V' = V$ et $S' = S$.

Fin.

Théorème 1: L'algorithme précèdent génère une grammaire G'<X,V', P', S'> équivalente à la grammaire de départ G<X,V,P, S>.

Démonstration : pour démontrer que L(G) = L(G'), la proposition suivante doit être démontrée par récurrence sur la longueur de w :

$$A \overset{*}{\underset{G'}{\vdash}} w \text{ ssi } w \neq \varepsilon \text{ et } A \overset{*}{\underset{G}{\vdash}} w.$$

La démonstration est laissée au lecteur.

2.7 Elimination des règles de productions de type A → B (enchaînement de variables).

L'algorithme suivant permet de construire à partir d'une grammaire G<X,V,P,S> ε-libre contenant des productions de type $A \to B$, une grammaires G'<X,V',P',S'> sans de telles productions.

Algorithme éliminer (In G ε-libre, out G'<X,V',P', S'> sans production de type A→B)

Début

1. Construire pour chaque non terminal A de V, l'ensemble $V_A = \{B \ / \ A \overset{*}{\underset{G}{\vdash}} B\}$ comme suit

- $V_0 = \{A\}$, $i = 1$
- $V_i = \{C \ / \ B \to C \in P \text{ et } B \in V_{i-1}\} \cup V_{i-1}$
- **Si** $V_i \neq V_{i-1}$ **alors** $i = i + 1$ et répéter l'étape précédente. **Sinon** $V_A = V_i$

2. Construire P' de la manière suivante :

Pour chaque A tel que $B \in V_A$:

Si $B \to \alpha \in P$ et $|\alpha| \neq 1$ **alors** rajouter $A \to \alpha$ à P'

Fin Pour

Fin

2.8 Grammaire sans cycle

Une grammaire algébrique G<X, V, P, S> est sans cycle si pour tout A de V il n'existe aucune dérivation de la forme $A \overset{+}{\underset{G}{\vdash}} A$.

2.9 Grammaire algébrique propre

Une grammaire à contexte libre G<X,V,P,S> est dite propre si elle est réduite, , ε-libre et sans cycle.

2.10 Forme Normale de Chomsky

a. Définition : Une grammaire à contexte libre G<X,V,P,S> est sous la forme normale de Chomsky (FNC) si toutes les productions de P sont de la forme :

1. $A \rightarrow BC$, $A, B, C \in V$
2. $A \rightarrow a$, $a \in X$

b. Proposition : A toute grammaire algébrique G<X,V,P,S>, il existe une grammaire G'<X',V',P',S'> sous forme normale de Chomsky équivalente à G.

Passage d'une grammaire à contexte libre vers une grammaire sous forme FNC :

Algorithme FNC (in G GCL propre sans enchaînement de variables, out G'<X,V',P', S'> sous forme FNC)

Début

V' = V, S' = S

Construire l'ensemble P' de la manière suivante :

1. Pour toutes les productions de P de type $A \rightarrow \alpha_1\ \alpha_2 \ldots \alpha_n \in P$ avec $n > 1$, $\alpha \in (X \cup V)$:

Si $\alpha_i \in X$, remplacer α_i par B_i dans la production, rajouter $B_i \rightarrow \alpha_i$ à P' et B_i à V'

La production obtenue est $A \rightarrow B_1\ B_2 \ldots B_n$

2. Rajouter à P' toutes les productions de type $(A \rightarrow a) \in P$, $A \in V$ et $a \in X$

3. Rajouter à P' toutes les productions de type $(A \rightarrow BC) \in P$, $A, B, C \in V$

4. Si $S \rightarrow \varepsilon \in P$, rajouter $S \rightarrow \varepsilon$ à P'

5. Pour toutes les productions de type $A \rightarrow B_1\ B_2 \ldots B_n$ dans P où $n \geq 3$, rajouter à P' les règles de production suivantes :

$A \rightarrow B_1\ X_1$

$X_1 \rightarrow B_2\ X_2$

…

$X_{n-2} \rightarrow B_{n-1}\ B_n$

Rajouter X_i à V $1 \leq i \leq n-2$

Fin

Exercice : Soit G<X, V, P, S> la grammaire propre définie ci-dessous où P=

{ $S \rightarrow aAB$ / BA
$A \rightarrow BBB$ / a
$B \rightarrow AS$ / b }

Donner la forme normale de Chomsky.

2.11 Forme Normale de Greibach :

Définition : Une grammaire à contexte libre G<X,V,P,S> est sous forme normale de Greibach (FNG) si toutes les productions de P sont de la forme :

1. $A \rightarrow a A_1 A_2 \dots A_n$ $a \in X$ et $A_1, A_2, \dots A_n \in V$ et $n \geq 1$
2. $A \rightarrow a$

Proposition : A toute grammaire algébrique G<X,V,P,S>, il existe une grammaire G'<X',V',P',S'> sous forme normale de Greibach équivalente à G.

9.2.1 Récursivité

Le problème du passage d'une grammaire à sa forme normale de Greibach (FNG) est la récursivité gauche qui peut-être directe ou indirecte. Pour pouvoir trouver la forme normale de Greibach, il faut transformer la récursivité gauche en récursivité droite.

a. Récursivité Directe

Définition 1 : On dit qu'une grammaire algébrique G<X,V, P,S> est récursive s'il existe dans P une production de type : $A \rightarrow \alpha A \beta$ avec $A \in V$, α et $\beta \in (X \cup V)^*$. A est dit récursive.

Définition 2 : On dit qu'une grammaire algébrique G<X,V,P,S> est récursive à gauche s'il existe dans P une production de type : $A \rightarrow A \beta$ avec $A \in V$, $\beta \in (X \cup V)^*$. A est dit récursive à gauche.

Définition 3 : On dit qu'une grammaire algébrique G<X,V,P,S> est récursive à droite s'il existe dans P une production de type : $A \rightarrow \alpha A$ avec $A \in V$, α et $\beta \in (X \cup V)^*$. A est récursive à droite.

Lemme 1: A toute grammaire récursive à gauche G<X,V,P,S>, il existe une grammaire récursive à droite équivalente G<X,V',P',S'>.

Construction de la grammaire récursive à droite G'<X',V',P',S'> à partir de la grammaire récursive à gauche G<X,V,P,S> :

Début
X' = X, S' = S, V'=V
Pour toutes les productions de P faire
Si $A \rightarrow A\ \alpha_1 / A\ \alpha_2 / \dots / A\ \alpha_n / \beta_1 / \beta_2 / \dots / \beta_m \in$ **P** avec α_i et $\beta_j \in (X \cup V)^*$ β_j ne commence pas par A
alors Rajouter à P' les productions :
$A \rightarrow \beta_1 / \beta_2 / \dots / \beta_m / \beta_1 A' / \beta_2 A' / \dots / \beta_m A' \in$ **P'**
$A' \rightarrow \alpha_1 A' / \alpha_2 A' / \dots / \alpha_n A' \in$ **P'** et $A' \in$ **V'**
Si $A \rightarrow \beta_1 / \beta_2 / \dots / \beta_m \in$ **P** avec $\beta_j \in (X \cup V)^*$ et β_j ne commence pas par A **alors**
$A \rightarrow \beta_1 / \beta_2 / \dots / \beta_m \in$ **P'**.
Fin pour
Fin

Lemme :

Soit G<X, V, P,S> une grammaire non récursive à gauche. Il existe un ordre partiel (linéaire) sur V tel que : si $A \rightarrow B\alpha \in P$ alors $A < B$.

Exercice : Donner l'algorithme de passage d'une grammaire G<X, V, P,S> algébrique à une grammaire algébrique G<X', V', P',S'> sous forme FNG.

3. Grammaire algébrique et Automate à pile

Proposition :

A toute grammaire algébrique G<X,V,P,S>, il existe un automate à pile vide $A_{p\varepsilon}$<X,Y,S,S_0,F,II, #> équivalent.

Démonstration :

Construction d'un automate à pile vide à partir d'une grammaire algébrique :

En entrée : La grammaire G<X, V, P, S> sans variables non productives et inaccessibles.

En sortie : l'automate à Pile vide A_ε<X, Y, S_A, S_0, F, II, #>.

Une grammaire ne donne pas la structure des mots du langage qu'elle reconnait mais les règles de production qui permettent de générer les mots. Ces mots sont obtenus en générant leur arbre de dérivation. Pour reconnaître un mot par cet automate, il faut générer son arbre de dérivation dans la pile.

Définition des paramètres de A_ε:

$Y = X \cup V \cup \{\#\}$ (puisque l'on va générer l'arbre de dérivation du mot à reconnaitre dans la pile on doit pouvoir empiler les éléments de X et V)

$S_A = \{S_0, S_1\}$ Deux états sont suffisant pour définir l'automate à pile de n'importe quelle grammaire algénrique

$F = \{S_1\}$

Procédure de construction de II à partir de P :

1. $II = \{ \# S_0 \rightarrow \# S S_1\}$ /* La première instruction de l'automate est l'empilement de l'axiome pour que l'on puisse générer l'arbre de dérivation dans la pile. On rajoute cette instruction quelque soit la grammaire que l'on transforme en automate. Vous remarquerez que cet automate ne fonctionne pas de la même manière que le premier que nous avons défini. Il n'y a pas toujours lecture de la lettre en entrée.

2. Pour toutes les productions de P ($A \rightarrow \alpha$) faire :

$II = II \cup \{ AS_1 \rightarrow \alpha^R S_1\}$ /* Ces instructions permettent de générer l'arbre de dérivation du mot à reconnaître dans la pile.

3. Pour tous les éléments x_i de X faire :

$II = II \cup \{ x_i S_1 x_i \rightarrow S_1\}$.

Exemple:

Soit G<X, V, P, S> une grammaire où P est défini comme suit :

P : { $S \rightarrow AB$

$A \rightarrow aAb$ / ε

$B \rightarrow cBd$ / ε }

L'automate à Pile vide A_ε<X, Y, S_A, S_0, F, II, #> reconnaissant L(G) est le suivant :

$Y = X \cup V \cup \{\#\}$

$S_A = \{S_0, S_1\}$

$F = \{S_1\}$

$II = \{\# S_0 \rightarrow \# S S_1$

$SS_1 \rightarrow BA\ S_1$

$AS_1 \rightarrow bAa\ S_1$

$AS_1 \rightarrow S_1$
$BS_1 \rightarrow dBc\ S_1$
$BS_1 \rightarrow S_1$
$a\ S_1\ a \rightarrow S_1$
$b\ S_1\ b \rightarrow S_1$
$\mathbf{c\ S_1\ c \rightarrow S_1}$
$\mathbf{d\ S_1\ d \rightarrow S_1}$
}

Reconnaissance d'un mot :

1. W= aabbcd $\in L(A_\varepsilon)$?

S_0aabbcd |— #SS_1aabbcd |— # BAS_1 aabbcd |— # $BbAaS_1$ aabbcd |— # $BbAS_1$ aabbcd |— # $BbbAaS_1$abbcd |— # $BbbAS_1$bbcd |— # $BbbS_1$bbcd |— # BbS_1bcd |— # BS_1cd |— # dBc S_1cd |— # dB S_1d |— # dS_1d |— # S_1

W= aabbcd fait passer l'automate d'une configuration initiale à une configuration finale alors w $\in L(A_\varepsilon)$.

2. W= aacd $\in L(A_\varepsilon)$

S_0aacd |— #SS_1aacd |— # BAS_1 aacd |— # $BbAaS_1$ aacd |— # $BbAS_1$acd |— # $BbbAaS_1$acd |— # $BbbAS_1$cd |— # $BbbS_1$cd A ce niveau, l'automate se bloque car il n'existe, dans II, aucune instruction dont le membre gauche est bS_1c.

W= aacd ne fait pas passer l'automate d'une configuration initiale à une configuration finale alors w $\notin L(A_\varepsilon)$.