Plan

- Motivation
- Langages et grammaires hors-contextes
- Forme normale de Chomsky
- L'algorithme de Cocke Kasami Younger
- Les grammaires hors contexte probabilistes
- Calcul de la probabilité d'une phrase
- Construction de l'arbre le plus probable
- Estimation des probabilités de la grammaire
- Limites des grammaires hors contexte probabilistes

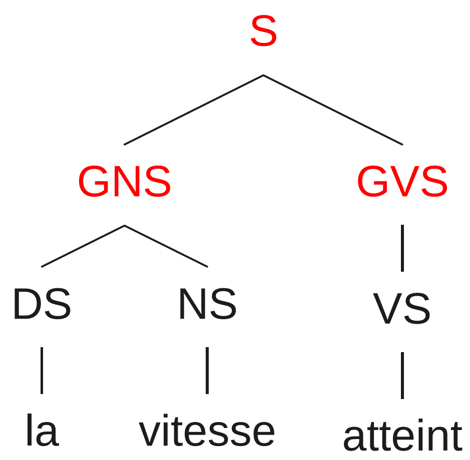
Limites des modèles n-gram

.

- la vitesse atteint . . .
- la vitesse des vagues atteint . . .
- la vitesse des vagues sismiques atteint . . .
- la vitesse des grandes vagues sismiques atteint . . .

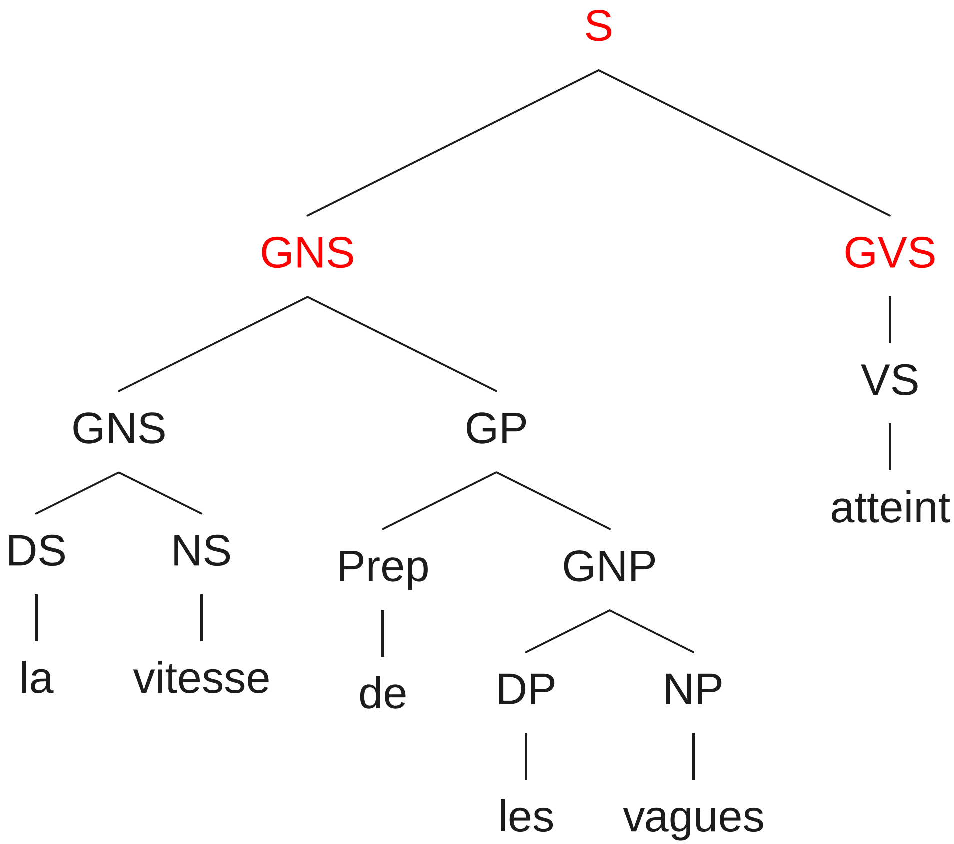
Structure syntaxique de la phrase

.



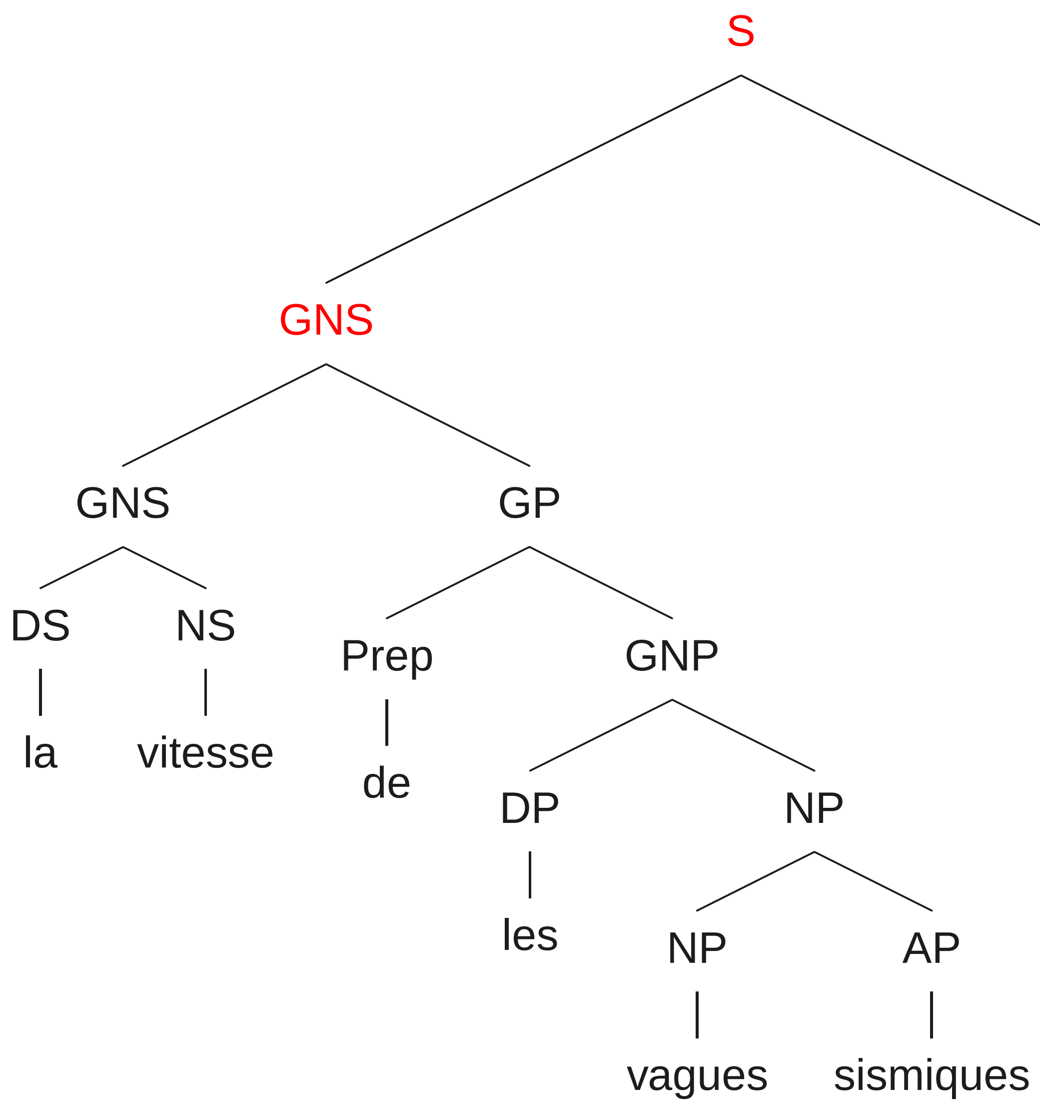
Structure syntaxique de la phrase

.



Structure syntaxique de la phrase

.



Grammaire probabiliste vue comme un modèle de langage .

- L'accord entre le sujet et le verbe est modélisé par une même règle dans les différentes phrases.
- La règle est insensible à la longueur du groupe nominal
- La grammaire peut facilement attribuer une meilleure probabilité à la phrase correcte

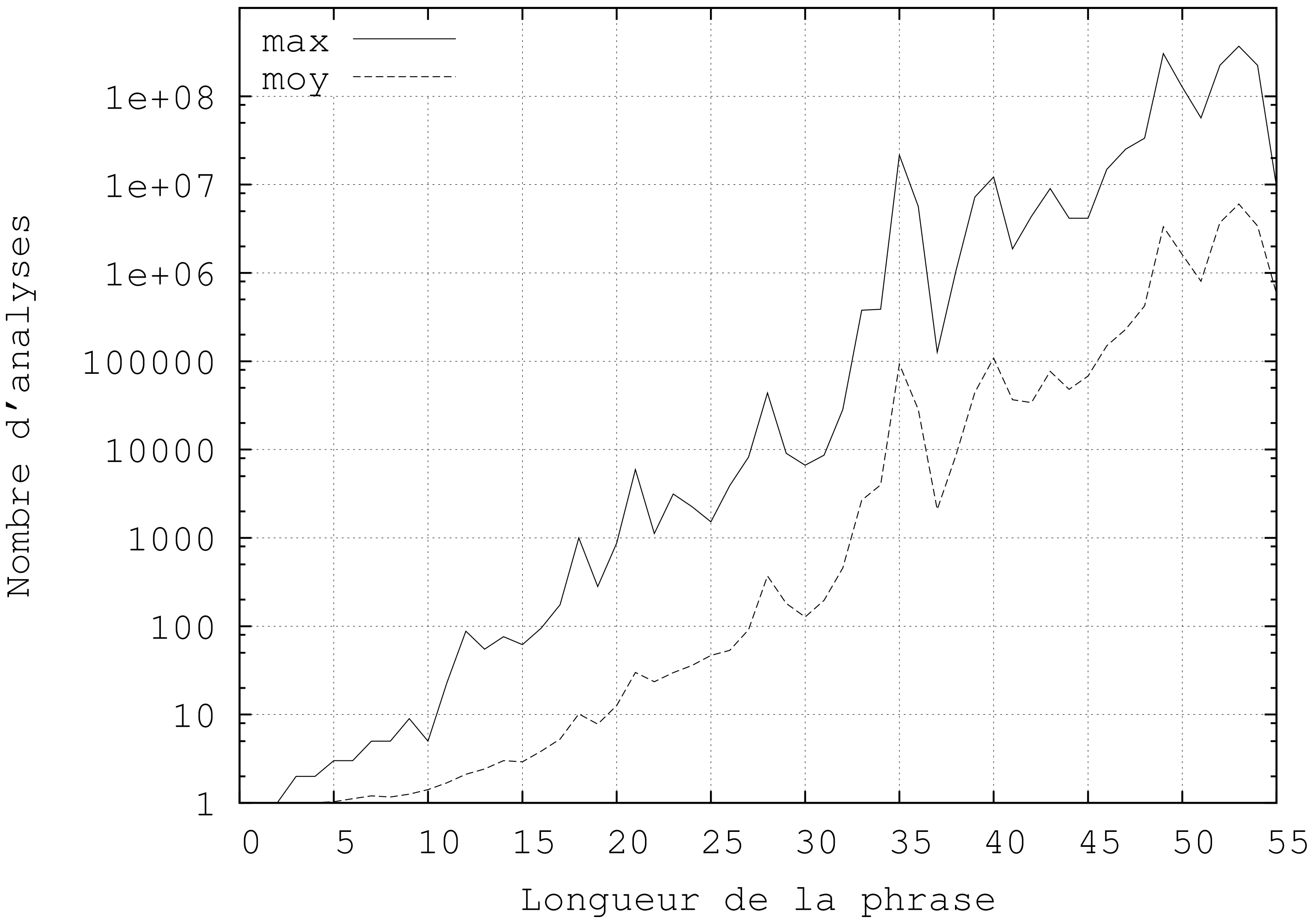
$$P(S \rightarrow GNS \;\; GVS) > P(S \rightarrow GNS \;\; GVP)$$

- La grammaire probabiliste peut être utilisée comme modèle de langage.
- Elle permet de calculer la probabilité de toute phrase $S \in L(G)$

$$P(S|G)$$

Désambiguïsation syntaxique

.



rammaire proba. vue comme un modèle de désambiguïsation
.

- Parmi les analyses automatiques d'une phrase, la très grande majorité est aberrante.
- On aimerait pouvoir classer les analyses selon leur plausibilité.
- La grammaire probabiliste peut être utilisée comme modèle de désambiguïsation
- Elle permet de calculer la probabilité d'un arbre T étant donné une phrase S :

$$P(T|S,G)$$

- et de retrouver l'arbre le plus probable :

$$\hat{T} = \arg\max_{T} P(T|S,G)$$

Notions de base de la théorie des langages

.

- Les symboles sont des éléments indivisibles qui vont servir de briques de base pour construire des mots.
- Un *alphabet* est un *ensemble* fini de symboles. On désigne conventionnellement un alphabet par la lettre grecque Σ.
- Une suite de symboles, appartenant à un alphabet Σ, mis bout à bout est appelé un *mot* (ou une *chaîne*) sur Σ. Le mot de longueur zéro est noté ε.
- L'ensemble de tous les mots que l'on peut construire sur un alphabet Σ est noté Σ^*.
- Un *langage* sur un alphabet Σ est un ensemble de mots construits sur Σ. Tout langage défini sur Σ est donc une partie de Σ^*.

Grammaires de réécriture

.

Une grammaire de réécriture est un 4-uplet $\langle N, \Sigma, P, S\rangle$ où :

- N est un ensemble de symboles non terminaux, appelé l'alphabet non terminal.
- Σ est un ensemble de symboles terminaux, appelé l'alphabet terminal, tel que N et Σ soient disjoints.
- P est un sous ensemble fini de :

$$(N \cup \Sigma)^* N (N \cup \Sigma)^* \times (N \cup \Sigma)^*$$

  un élément (α, β) de P, que l'on note $\alpha \rightarrow \beta$ est appelé une règle de production ou règle de réécriture.
- S est un élément de N appelé l'*axiome* de la grammaire.

Proto-phrases d'une grammaire

.

Les proto-phrases d'une grammaire $G = \langle N, \Sigma, P, S \rangle$ sont des mots construits sur l'alphabet $\Sigma \cup N$, on les définit récursivement de la façon suivante :

- S est une proto-phrase de G
- si $\alpha\beta\gamma$ est une proto-phrase de G et $\beta \rightarrow \delta \in P$ alors $\alpha\delta\gamma$ est une proto-phrase de G.

Une proto-phrase de G ne contenant aucun symbole non terminal est appelé un mot généré par G. Le *langage généré par* G, noté $L(G)$ est l'ensemble des mots générés par G.

Dérivation

.

- L'opération qui consiste à générer une proto-phrase $\alpha\delta\gamma$ à partir d'une proto-phrase $\alpha\beta\gamma$ et d'une règle de production r de la forme $\beta \rightarrow \delta$ est appelée l'opération de *dérivation*. Elle se note à l'aide d'une double flèche :

$$\alpha\beta\gamma \Rightarrow \alpha\delta\gamma$$

- On note $\alpha \stackrel{k}{\Rightarrow} \beta$ pour indiquer que β se dérive de α en k étapes.
- On définit aussi les deux notations $\stackrel{+}{\Rightarrow}$ et $\stackrel{*}{\Rightarrow}$ de la façon suivante :
  - $\alpha \stackrel{+}{\Rightarrow} \beta \equiv \alpha \stackrel{k}{\Rightarrow} \beta$ avec $k > 0$
  - $\alpha \stackrel{*}{\Rightarrow} \beta \equiv \alpha \stackrel{k}{\Rightarrow} \beta$ avec $k \geq 0$

Dérivation

.

- $L(G)$ est défini de la façon suivante :

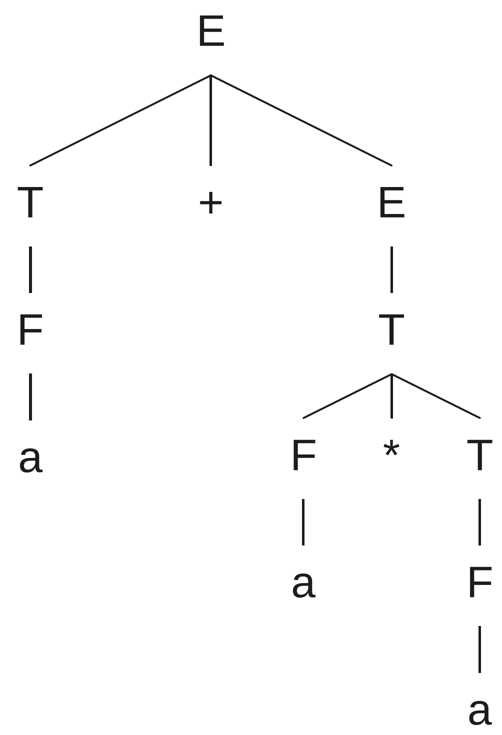
$$L(G) = \{m \in \Sigma^* | S \stackrel{*}{\Rightarrow} m\}$$

- Deux grammaires G et G' sont équivalentes si les langages $L(G)$ et $L(G')$ sont identiques.

Sens de dérivation

- Les proto-phrases générées lors d'une dérivation peuvent comporter plus d'un symbole non terminal :
$\underline{E} \Rightarrow T + \underline{E} \Rightarrow \underline{T} + T \Rightarrow F + \underline{T} \Rightarrow F + \underline{F} * T \Rightarrow F + a * \underline{T} \Rightarrow \underline{F} + a * F \Rightarrow a + a * \underline{F} \Rightarrow a + a * a$
- Dérivation droite : on réécrit le non terminal le plus à droite :
$\underline{E} \Rightarrow T + \underline{E} \Rightarrow T + \underline{T} \Rightarrow T + F * \underline{T} \Rightarrow T + F * \underline{F} \Rightarrow T + \underline{F} * a \Rightarrow \underline{T} + a * a \Rightarrow \underline{F} + a * a \Rightarrow a + a * a$
- Dérivation gauche : on réécrit le non terminal le plus à gauche :
$\underline{E} \Rightarrow \underline{T} + E \Rightarrow \underline{F} + E \Rightarrow a + \underline{E} \Rightarrow a + \underline{T} \Rightarrow a + \underline{F} * T \Rightarrow a + a * \underline{T} \Rightarrow a + a * \underline{F} \Rightarrow a + a * a$

Arbre de dérivation



Un arbre de dérivation pour G ($G = \langle N, \Sigma, P, S\rangle$) est un arbre ordonné et étiqueté dont les étiquettes appartiennent à l'ensemble $N \cup \Sigma \cup \{\varepsilon\}$. Si un nœud de l'arbre est étiqueté par le non terminal A et ses fils sont étiquetés $X_1, X_2, ..., X_n$ alors la règle $A \rightarrow X_1, X_2, ..., X_n$ appartient à P.

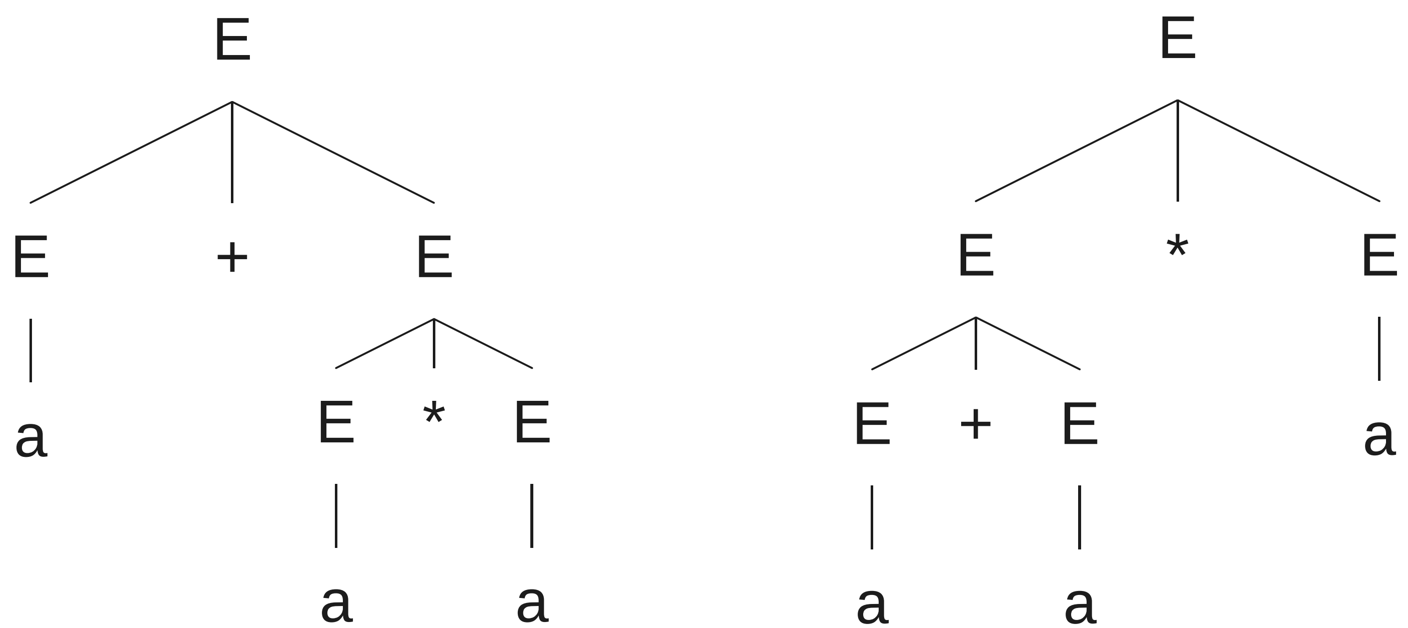
Arbre de dérivation

.

- Un arbre de dérivation indique les règles qui ont été utilisées dans une dérivation, mais pas l'ordre dans lequel elles ont été utilisées.
- A un arbre de dérivation correspondent une seule dérivation droite et une seule dérivation gauche.

Ambiguïté

Une grammaire G est **ambigüe** s'il existe au moins une chaîne c dans $L(G)$ à laquelle correspond plus d'un arbre de dérivation.
Exemple : $E \rightarrow E + E|E * E|a$



Types de règles

.

Les grammaires peuvent être classées en fonction de la forme de leurs règles de production. On définit cinq types de règles de production :

- règles régulières à gauche :
  Une règle est régulière à gauche si et seulement si elle est de la forme $A \rightarrow xB$ ou $A \rightarrow x$ avec $A, B \in N$ et $x \in \Sigma^*$.
- règles régulières à droite :
  Une règle est régulière à gauche si et seulement si elle est de la forme $A \rightarrow Bx$ ou $A \rightarrow x$ avec $A, B \in N$ et $x \in \Sigma^*$.
- règles hors-contexte :
  Une règle $A \rightarrow \alpha$ est un règle hors-contexte si et seulement si : $A \in N$ et $\alpha \in (N \cup \Sigma)^*$

Types de règles

.

- règles contextuelles :
  Une règle $\alpha \rightarrow \beta$ est une règle contextuelle si et seulement si : $\alpha = gAd$ et $\beta = gBd$ avec $g, d, B \in (N \cup \Sigma)^*$ et $A \in N$ le nom “contextuelle” provient du fait que A se réecrit B uniquement dans le contexte g_d.
- règles sans restrictions Une règle $\alpha \rightarrow \beta$ est une règle sans restriction si et seulement si : $|\alpha| \geq 1$

Type d'une grammaire

.

une grammaire G est :

- *régulière* si elle est régulière à droite ou régulière à gauche. Une grammaire est régulière à gauche si *toutes* ses règles sont régulières à gauche et une grammaire est régulière à droite si *toutes* ses règles sont régulières à droite.
- *hors contexte* si toutes ses règles de production sont hors contexte.
- *dépendante du contexte* si toutes ses règles de production sont dépendantes du contexte.
- *sans restrictions* si toutes ses règles de production sont sans restrictions.

Les types de grammaires définis ci-dessus forment une hiérarchie appelée *hiérarchie de Chomsky*

Type d'un langage

.

Un langage pouvant être généré par une grammaire de type x et pas par une grammaire d'un type supérieur dans la hiérarchie, est appelé un langage de type x.

Exemples

.

- langages réguliers
  $L = a^n$, avec $n \geq 0$
  $G = \langle \{S\}, \{a\}, \{S \rightarrow aS|\varepsilon\}, S\rangle$
- langages hors contextes
  $L = a^n b^n$, avec $n \geq 0$
  $G = \langle \{S\}, \{a, b\}, \{S \rightarrow aSb|\varepsilon\}, S\rangle$

  $L = mm$ avec $m \in \Sigma^*$ (langage mirroir)
  $G = \langle \{S\}, \{a, b\}, \{S \rightarrow aSa|bSb|aa|bb\}, S\rangle$
- langages contextuels
  $L = a^n b^n c^n$, avec $n \geq 0$
  $G = \langle \{S, S_1, S_2\}, \{a, b, c\}, \{S \rightarrow aS_1c, S_1 \rightarrow b|SS_2, cS_2 \rightarrow S_2c, bS_2 \rightarrow bb\}, S\rangle$.

Forme normale de Chomsky

.

- Une grammaire hors-contexte est en forme normale de Chomsky si toutes ses règles sont de la forme :

  $$A \rightarrow BC \text{ ou } A \rightarrow a$$

  avec $A, B \in N$ et $a \in \Sigma$. De plus, on autorise la règle $S \rightarrow \varepsilon$ si S est l'axiome de la grammaire et s'il n'apparaît jamais dans la partie droite d'une règle.
- Tout langage hors-contexte peut être généré par une grammaire hors-contexte en forme normale de Chomsky.

Conversion en forme normale de Chomsky

.

1. Création d'un nouvel axiome. On crée un nouveau symbole S_0 et on ajoute la règle $S_0 \to S$. Cette modification garantit que l'axiome n'apparaît pas dans une partie droite de règle.
2. Elimination des règles-ε. On élimine une règle de la forme $A \to \varepsilon \in P$, pour $A \neq S_0$ puis, pour toute occurrence de A dans une règle de P, on ajoute une nouvelle règle dans laquelle cette occurrence de A a été éliminée. La règle $X \to \alpha A \beta A \gamma$, par exemple, provoquera l'ajout des trois règles suivantes : $X \to \alpha\beta A\gamma$, $X \to \alpha A \beta\gamma$ et $X \to \alpha\beta\gamma$. Si $X \to A \in P$ alors on ajoute $X \to \varepsilon$ à moins que cette dernière n'ait déjà été éliminée. On recommence cette étape tant que P possède des règles-ε.

Conversion en forme normale de Chomsky

.

1. Elimination des règles $A \rightarrow B$. On élimine une règle de la forme $A \rightarrow B$. Pour toute règle de la forme $B \rightarrow \alpha$, on ajoute une règle $A \rightarrow \alpha$ à moins qu'il ne s'agisse d'une règle déjà éliminée. On recommence cette étape tant que P possède des règles de la forme $A \rightarrow B$.
2. Transformation des règles restantes. Toute règle de la forme $A \rightarrow \alpha_1\alpha_2 \dots \alpha_k$ avec $k \geq 3$ et $\alpha_i \in \Sigma \cup N$, est remplacée par les règles $A \rightarrow \alpha_1 A_1$, $A_1 \rightarrow \alpha_2 A_2$, ..., $A_{k-2} \rightarrow \alpha_{k-1}\alpha_k$ où $A_1 \dots A_k$ sont de nouveaux non terminaux. Si $k \geq 2$, on remplace tout symbole terminal α_i des règles précédentes par un nouveau symbole non terminal U_i et on ajoute la règle $U_i \rightarrow \alpha_i$

Exemple

.

1. Création d'un nouvel axiome.

$$S \rightarrow ASA|aB$$
$$A \rightarrow B|S$$
$$B \rightarrow b|\varepsilon$$

$$S_0 \rightarrow S$$
$$S \rightarrow ASA|aB$$
$$A \rightarrow B|S$$
$$B \rightarrow b|\varepsilon$$

2. Elimination des règles-ε.

$$S_0 \rightarrow S$$
$$S \rightarrow ASA|aB|a$$
$$A \rightarrow B|S|\varepsilon$$
$$B \rightarrow b$$

$$S_0 \rightarrow S$$
$$S \rightarrow ASA|aB|a|SA|AS|S$$
$$A \rightarrow B|S$$
$$B \rightarrow b$$

Exemple

.

1. Elimination des règles $A \rightarrow B$.

   Elimination de $S \rightarrow S$ à gauche et de $S_0 \rightarrow S$ à droite :

$S_0 \rightarrow S$	$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$	$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow B \mid S$	$A \rightarrow B \mid S$
$B \rightarrow b$	$B \rightarrow b$

   Elimination de $A \rightarrow B$ à gauche et de $A \rightarrow S$ à droite :

$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$	$S_0 \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$	$S \rightarrow ASA \mid aB \mid a \mid SA \mid AS$
$A \rightarrow S \mid b$	$A \rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS$
$B \rightarrow b$	$B \rightarrow b$

Exemple

.

1. Transformation des règles restantes.

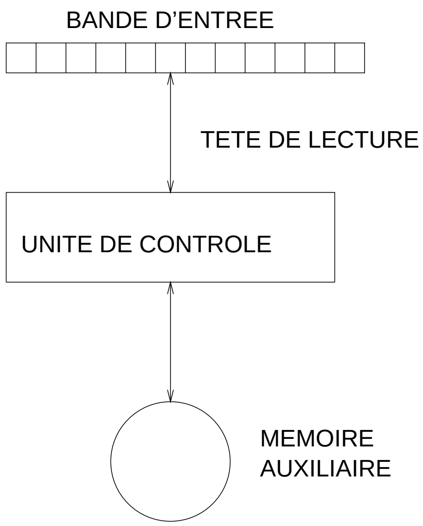
$$S_0 \rightarrow AA_1|UB|a|SA|AS$$
$$S \rightarrow AA_1|UB|a|SA|AS$$
$$A \rightarrow b|AA_1|UB|a|SA|AS$$
$$A_1 \rightarrow SA$$
$$U \rightarrow A$$
$$B \rightarrow b$$

Grammaire v/s Reconnaisseur

.

- Une **grammaire** d'un langage L permet de générer toutes les chaînes appartenant à L.
- Un **reconnaisseur** pour un langage L est un programme qui prend en entrée une chaîne c et répond *oui* si c appartient à L et *non* sinon.
- Pour chaque classe de grammaire, il existe une classe de reconnaisseurs qui définit la même classe de langages.
  - Grammaires régulières $\Longleftrightarrow$ Automates à états finis
  - Grammaires hors contexte $\Longleftrightarrow$ Automates à pile

Reconnaisseur



Configuration et mouvement

.

- **Configuration** d'un reconnaisseur :
  - Etat de l'unité de contrôle
  - Contenu de la bande d'entrée et position de la tête
  - Contenu de la mémoire
- **Mouvement** : passage d'une configuration à une autre ($C_1 \vdash C_2$)

L'unité de contrôle est dite **déterministe** si à toute configuration correspond au plus un mouvement. S'il peut exister plus d'un mouvement, elle est dite **non déterministe.**

Configurations

.

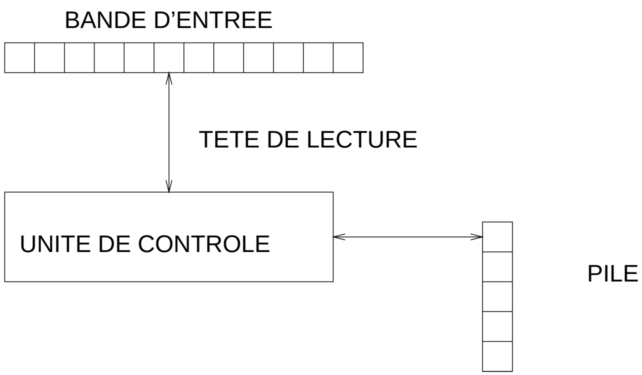
- **configuration initiale**
  - L'unité de contrôle est dans un état initial
  - La tête est au début de la bande
  - La mémoire contient un élément initial.
- **configuration d'acceptation**
  - L'unité de contrôle est dans un état d'acceptation
  - La tête de lecture est à la fin de la bande
  - La mémoire se trouve dans un état d'acceptation.

Reconnaissance

- Une chaîne w est **acceptée** par un reconnaisseur si, partant de l'état initial, avec w sur la bande d'entrée, le reconnaisseur peut faire une série de mouvements pour se retrouver dans un état d'acceptation.
- Le **langage accepté** par un reconnaisseur est l'ensemble de toutes les chaînes qu'il accepte.

Automates à pile

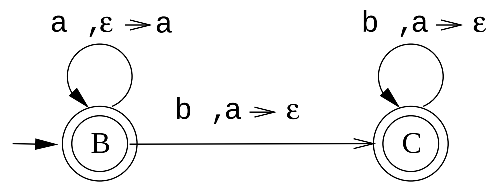
.



Une configuration d'un automate à pile est un triplet (q, w, α) où α représente le contenu de la pile, le symbole se trouvant au sommet de la pile est le symbole le plus à gauche.

Exemple : $L = \{a^n b^n \mid n \geq 0\}$

.



$$(B, aaabbb, \$) \vdash (B, aabbb, a\$) \vdash (B, abbb, aa\$) \vdash (B, bbb, aaa\$) \vdash$$
$$(C, bb, aa\$) \vdash (C, b, a\$) \vdash (C, \varepsilon, \$)$$

Automate à pile : définition

.

Un automate à pile est un septuplet $\langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F\rangle$

- Q est l'ensemble des états
- Σ est l'alphabet d'entrée
- Γ est l'alphabet de symboles de pile
- δ est la fonction de transition

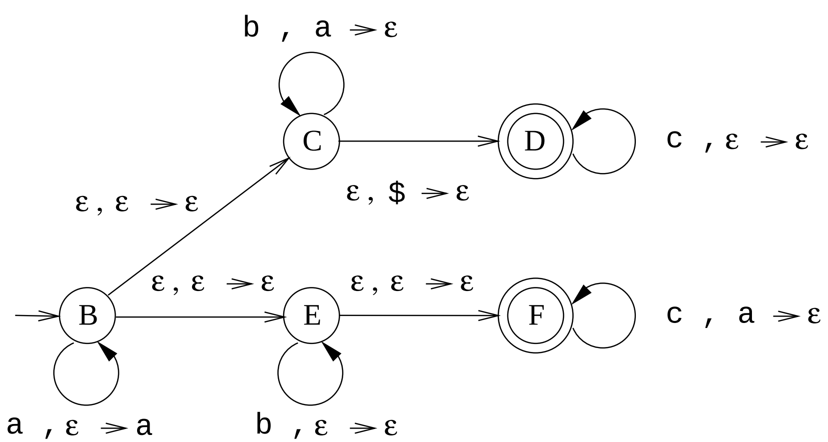
$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \wp(Q \times \Gamma^*)$$

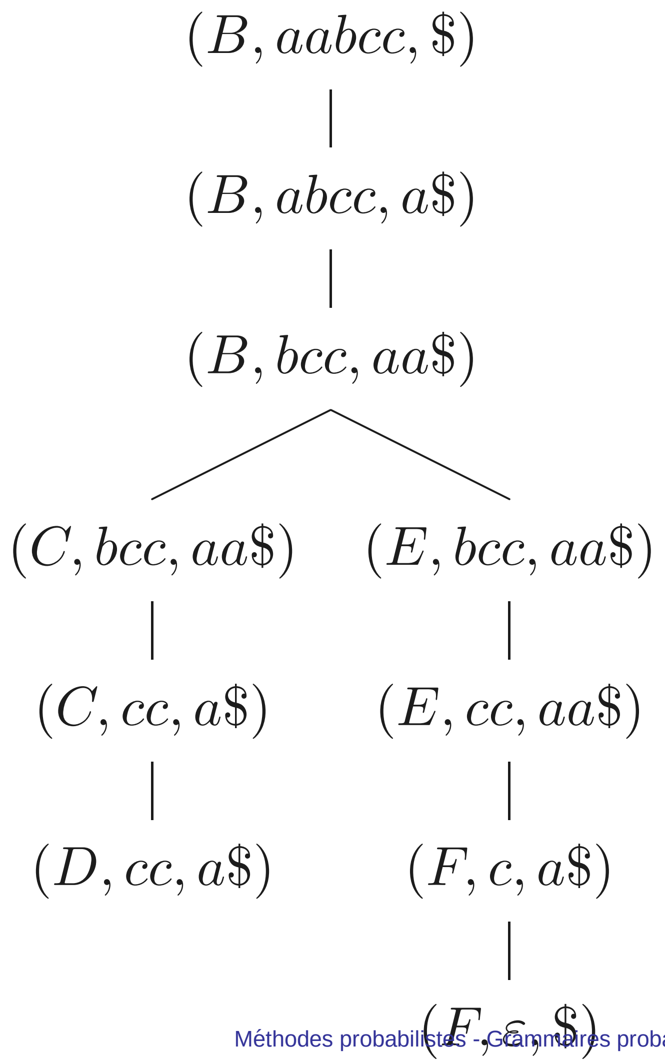
- $q_0 \in Q$ est l'état initial
- $Z_0 \in \Gamma$ est le symbole de fond de pile
- $F \subseteq Q$ est l'ensemble des états d'acceptation

Reconnaissance avec un AP non déterministe

.

$$L = \{a^i b^j c^k \mid i, j, k \geq 0 \ et \ i = j \ ou \ i = k\}$$

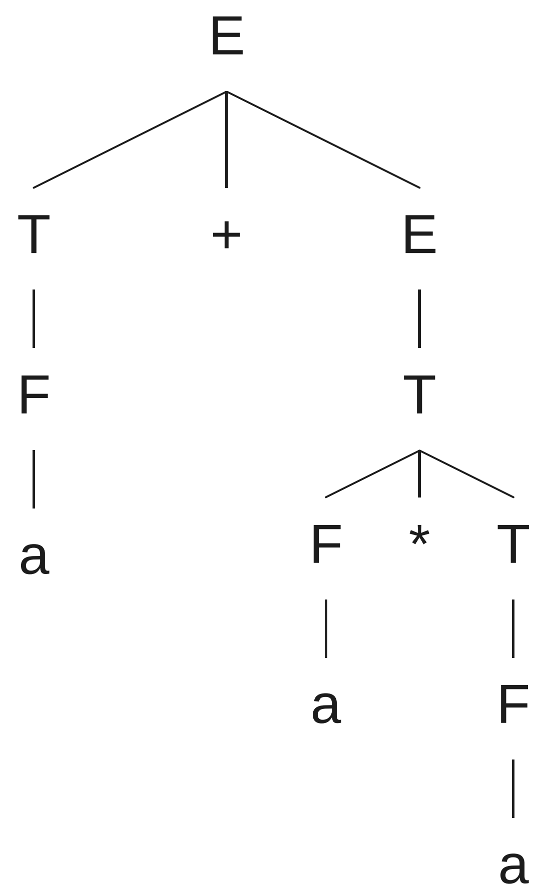




Analyse syntaxique

Etant donné $c \in \Sigma^*$ et $G = \langle \Sigma, N, P, A \rangle$, analyser c consiste à trouver pour c son (et éventuellement ses) arbre de dérivation.

$$
\begin{array}{lcl}
E & \longrightarrow & T + E | T \\
T & \longrightarrow & F * T | F \\
F & \longrightarrow & (E) | a
\end{array}
$$



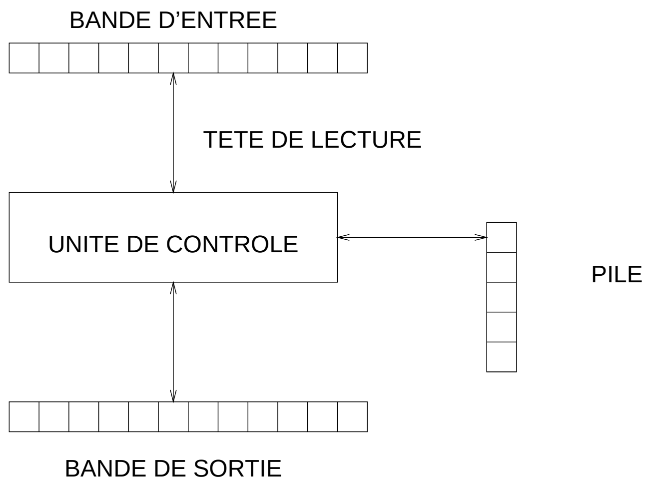
Sens d'analyse

.

- Analyse descendante
  Séquence de dérivations gauches à partir de l'axiome
  $E \Rightarrow T + E \Rightarrow F + E \Rightarrow a + E \Rightarrow a + T \Rightarrow a + F * T \Rightarrow a + a * T \Rightarrow a + a * F \Rightarrow a + a * a$
- Dérivation ascendante
  Séquence de dérivation telle que la séquence inverse soit une dérivation droite de c.
  $a + a * a \Leftarrow F + a * a \Leftarrow T + a * a \Leftarrow T + F * a \Leftarrow T + F * F \Leftarrow T + F * T \Leftarrow T + E \Leftarrow E$

Transducteurs à pile

.



Un transducteur à pile est un automate à pile qui **émet**, à chaque déplacement, un suite finie de symboles de sortie.
Une configuration d’un transducteur à pile est un quadruplet (q, w, α, y) y étant une séquence de symboles de sortie.

Transducteur à pile : définition

.

Un transducteur à pile est un 8-uplet $\langle Q, \Sigma, \Gamma, \Delta, \delta, q_0, Z_0, F\rangle$

- Q est l'ensemble des états
- Σ est l'alphabet d'entrée
- Γ est l'alphabet de symboles de pile
- Δ est l'alphabet de sortie
- δ est la fonction de transition

$$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \wp(Q \times \Gamma^* \times \Delta^*)$$

- $q_0 \in Q$ est l'état initial
- $Z_0 \in \Gamma$ est le symbole de fond de pile
- $F \subseteq Q$ est l'ensemble des états d'acceptation

Analyseur gauche

.

1 $E \rightarrow E + T$ 2 $E \rightarrow T$

3 $T \rightarrow T * F$ 4 $T \rightarrow F$

5 $F \rightarrow (E)$ 6 $F \rightarrow a$

- Dérivation gauche de $a * (a + a)$:
  $E \overset{2}{\Rightarrow} T \overset{3}{\Rightarrow} T * F \overset{4}{\Rightarrow} F * F \overset{6}{\Rightarrow} a * F \overset{*}{\Rightarrow} a * (a * a)$
- Analyse gauche : 23465124646

Analyseur gauche

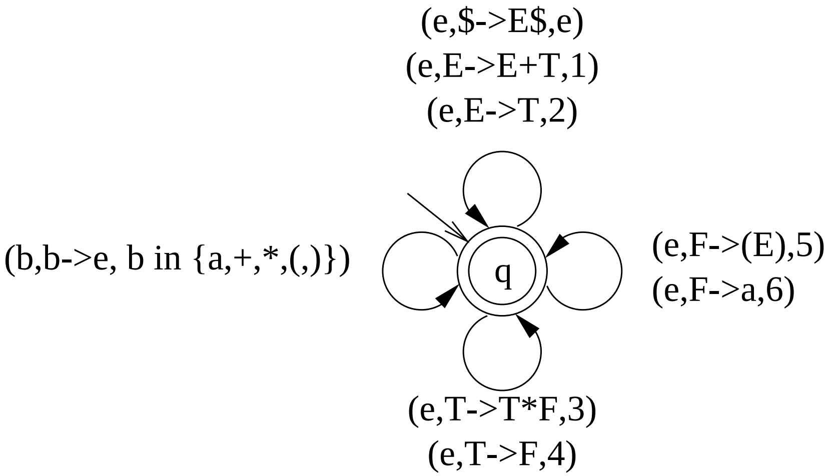
.

Soit une CFG G dont les règles ont été numérotées de 1 à p.
On appelle un **analyseur gauche** de G, un transducteur à pile non déterministe T_G^g qui produit pour une entrée w, une dérivation gauche de w.
Performances :

- Espace : $\mathcal{O}(|w|)$
- Temps : $\mathcal{O}(c^{|w|})$

Analyseur gauche : Exemple

.



$$
\begin{array}{ll}
 & (q, a+a*a, \$) \\
\vdash & (q, a+a*a, E\$) \\
\vdash & (q, a+a*a, E+T\$, 1) \\
\vdash & (q, a+a*a, T+T\$, 12) \\
\vdash & (q, a+a*a, F+T\$, 124) \\
\vdash & (q, a+a*a, a+T\$, 1246) \\
\vdash & (q, +a*a, +T\$, 1246) \\
\vdash & (q, a*a, T\$, 1246) \\
\vdash & (q, a*a, T*F\$, 12463) \\
\vdash & (q, a*a, F*F\$, 124634) \\
\vdash & (q, a*a, a*F\$, 1246346) \\
\vdash & (q, a, F\$, 1246346) \\
\vdash & (q, a, a\$, 12463466) \\
\vdash & (q, \varepsilon, \$, 12463466)
\end{array}
$$

Méthodes tabulaires

.

Programmation dynamique, les analyses partielles sont effectuées une seule fois et stockées dans une table.

Méthode	Espace	Temps
CYK	$\mathcal{O}(\|w\|^2)$	$\mathcal{O}(\|w\|^3)$
Earley	$\mathcal{O}(\|w\|^2)$	$\mathcal{O}(\|w\|^3)$

L'algorithme de Cocke-Younger-Kasami

.

Entrée:

- une grammaire hors-contexte sous forme normale de Chomsky sans ϵ-transitions.
- une chaine $w = a_1 a_2 \ldots a_n \in \Sigma^+$

Sortie:

- Une table d'analyse T pour w telle que la case $t_{i,j}$ contient A si et seulement si $A \stackrel{+}{\Rightarrow} a_i a_{i+1} \ldots a_j$

Exemple

.

- Grammaire initiale

$$E \longrightarrow F + E|F$$
$$F \longrightarrow T * F|T$$
$$T \longrightarrow (E)|a|b$$

- Grammaire sous forme normale de Chomsky

$$\begin{array}{lll} E \rightarrow YE|TN|KL|a|b & N \rightarrow ZF & Z \rightarrow * \\ F \rightarrow TN|KL|a|b & L \rightarrow EM & K \rightarrow (\\ T \rightarrow KL|a|b & V \rightarrow + & M \rightarrow) \\ Y \rightarrow FV & & \end{array}$$

Analyse de la chaine $(a + b) * b$

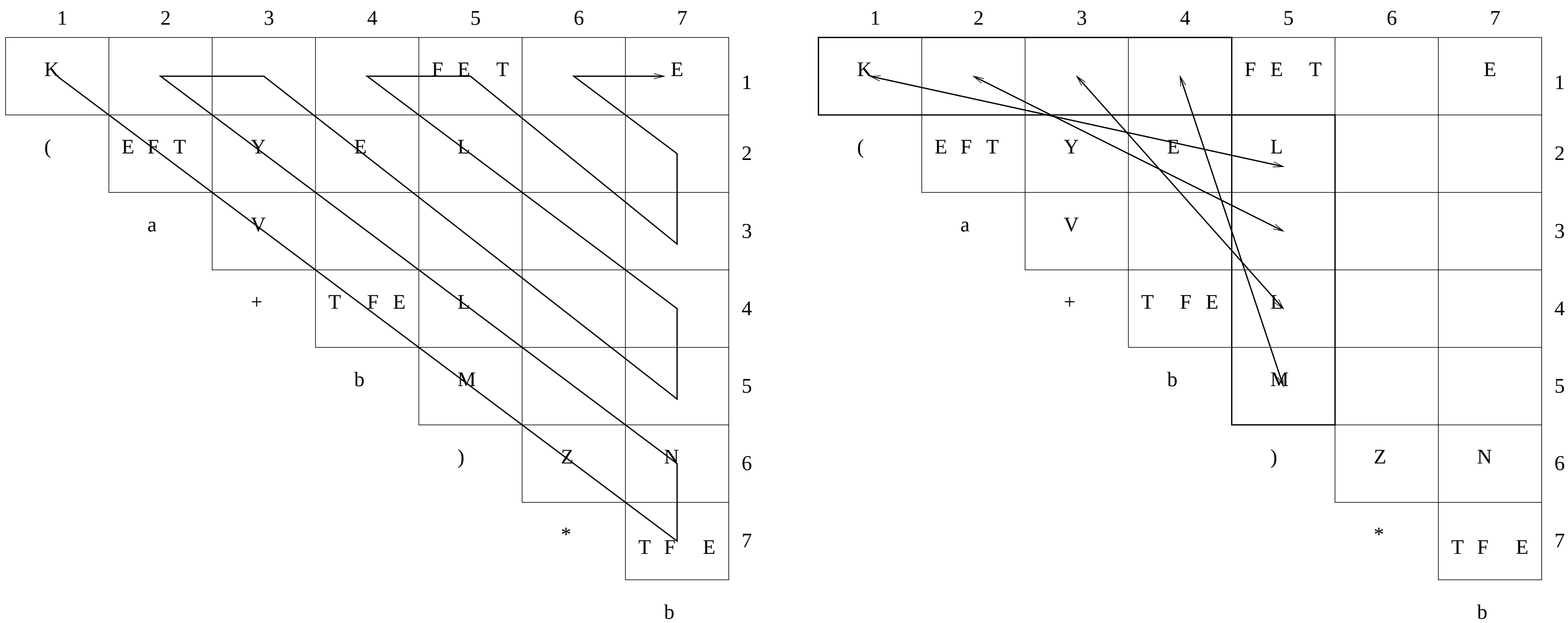
.

$$E \rightarrow YE|TN|KL|a|b \qquad N \rightarrow ZF \qquad Z \rightarrow *$$
$$F \rightarrow TN|KL|a|b \qquad L \rightarrow EM \qquad K \rightarrow ($$
$$T \rightarrow KL|a|b \qquad V \rightarrow + \qquad M \rightarrow)$$
$$Y \rightarrow FV$$

1	2	3	4	5	6	7	
K				F E T		E	1
(	E F T	Y	E	L			2
	a	V					3
		+	T F E	L			4
			b	M			5
				)	Z	N	6
					*	T F E	7
						b	

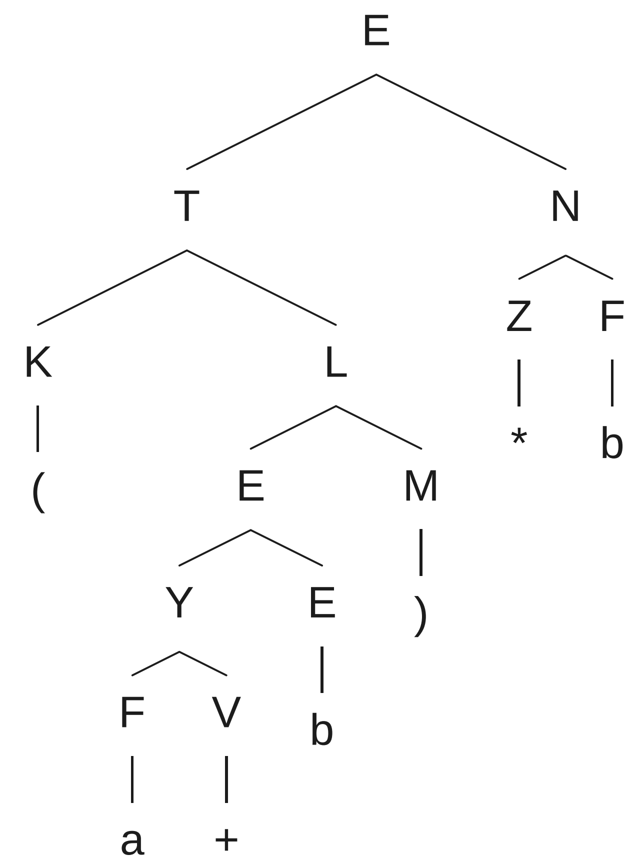
Analyse de la chaine $(a + b) * b$

.



Analyse de la chaine $(a+b)*b$

1	2	3	4	5	6	7	
K -> (				F -> K L E -> K L T -> K L		E -> T N	1
(	E -> a F -> a T -> a	Y -> FV	E -> Y E	L -> E M			2
	a	V -> +					3
		+	E -> b F -> b T -> b	L -> E M			4
			b	M ->)			5
				)	Z -> *	N -> Z F	6
					*	F -> b E -> b T -> b	7
						b	



Algorithme CYK

.

`pour` $i = 1$ `à` n `faire` $\{$ INITIALISATION $\}$

$t_{i,i} = \{A | A \rightarrow a_i\}$

`pour` $j = 1$ `à` n `faire`

`pour` $i = j - 1$ `à` 1 `faire`

`pour` $k = i$ `à` $j - 1$ `faire`

$t_{i,j} = t_{i,j} \cup \{A | A \rightarrow BC\}$

`avec` $B \in t_{i,k}$ `et` $C \in t_{k+1,j}$