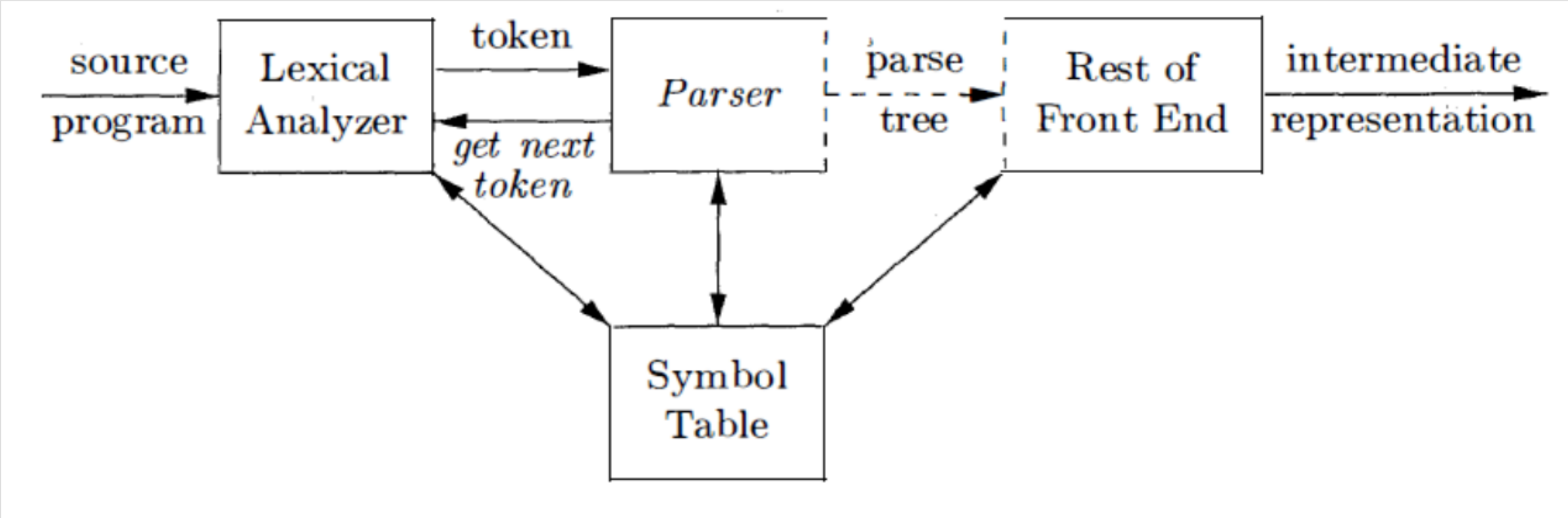
1- Introduction

- Si l'analyse lexicographique résout le problème de l'appartenance, des mots du texte source, au langage. L'analyse syntaxique a pour objectif de vérifier si la structure des phrases du texte source appartient au langage en question.
- L'analyse syntaxique ou parsing, consiste à déterminer la syntaxe ou la structure d'un programme.
- La syntaxe d'un langage de programmation est donnée par une grammaire hors-contexte.

- L'analyseur syntaxique reçoit une suite d'unités lexicales de la part de l'analyseur lexical et doit vérifier que cette suite peut être engendrée par la grammaire du langage.
- Cela consiste à construire un arbre syntaxique de la suite des unités lexicales formant le programme (le texte en entrée)

2- Position de l'analyseur syntaxique



3- Grammaire hors-contexte versus expression régulière

Les grammaires sont des notations plus fortes que les expressions régulières. Chaque construction pouvant être décrite par une expression régulière peut être décrite par une grammaire hors-contexte.

Pourquoi utiliser les expressions régulières pour définir le lexique d'un langage de programmation ?

- La séparation de la structurer syntaxique d'un langage en deux modules de tailles faciles à gérer,
- Les règles lexicales d'un langage sont généralement simple, dont leur description ne nécessite pas un mécanisme assez puissant tel que les grammaires,

- Les expressions régulières fournissent une notation plus concise et facile à comprendre pour les tokens que les grammaires
- Des analyseurs lexicaux plus efficaces peuvent être construits automatiquement à partir des expressions régulières qu'à partir des grammaires

Rappel

Une grammaire est dite ambiguïté si pour une chaine du texte source. Il existe au moins deux arbres syntaxiques générés par *les dérivations les plus gauches*.

- **Dérivation la plus à gauche** : est entièrement composée de dérivations en une étape dans lesquelles à chaque fois c'est le non-terminal le plus à gauche qui est remplacé. $\alpha \underset{lm}{\Rightarrow} {}^{*}\beta$

- **Dérivation la plus à droite :** à chaque étape de dérivation c'est le non-terminal le plus à droite qui est remplacé. $\alpha \underset{rm}{\Rightarrow} {}^{*}\beta$

4- Méthodes d'Analyse Syntaxique

Il existe deux méthodes d'analyse standards, formelles, et complètement maitrisées :

- L'analyse descendante (Top-Down Parsing)
- L'analyse ascendante (Bottom-Up Parsing)

- **Méthode d’analyse descendante :** Etant donner une grammaire et un texte source, partir de l’axiome et tenter de créer une chaine identique à la chaine d’entrée par dérivations successives. L’opération de base est le remplacement d’un symbole non terminal par une de ses parties droites. On dit qu’on père par génération (dérivation).

  Résultat : arbre syntaxique dont la racine de l’arbre est axiome, et les feuilles sont les unités lexicales

- **Méthode d'analyse ascendante**

  Cette méthode travaille de façon inverse de la $1^{ère}$ méthode. Elle consiste à trouver une chaine de symboles identique à une partie droite et la remplacer par le symbole non-terminal correspondant jusqu'à arriver a l'axiome. Dans ce cas on opère par réduction.

5- Analyse descendante

Deux méthodes : L'analyse descendante récursive, et l'analyse descendante prédictive.

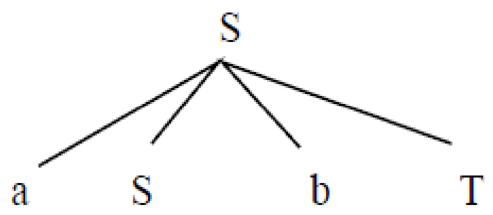
Exemple

$S \rightarrow aSbT \mid cT \mid d$

$T \rightarrow aT \mid bS \mid c$

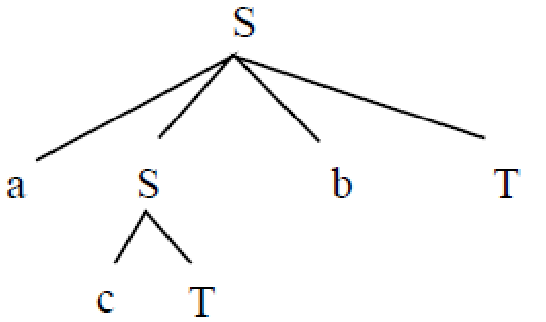
Avec le mot *w* = accbbadbc, on part avec l'arbre contenant le seul sommet *S*

La lecture de la première lettre du mot *a* nous permet d'avancer la construction



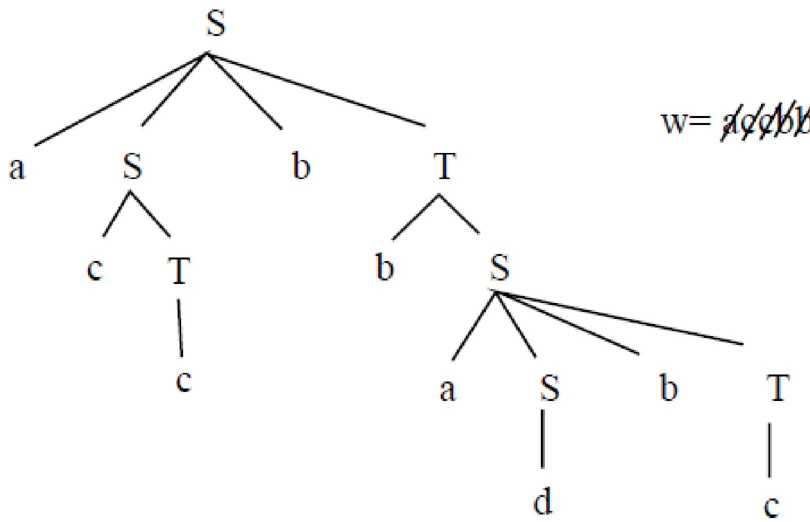
w= $\cancel{a}$ccbbadbc

Puis la deuxième lettre c nous amène à :



w= ~~ac~~cbbadbc

Et ainsi de suite jusqu'à :



w= ~~accbbadbc~~

Ainsi le mot *w* appartient au langage généré par la grammaire.

Sur cette exemple c'est très facile car chaque règle commence par un terminal différent, donc en sait immédiatement quelle règle utilisée.

Exemple 2

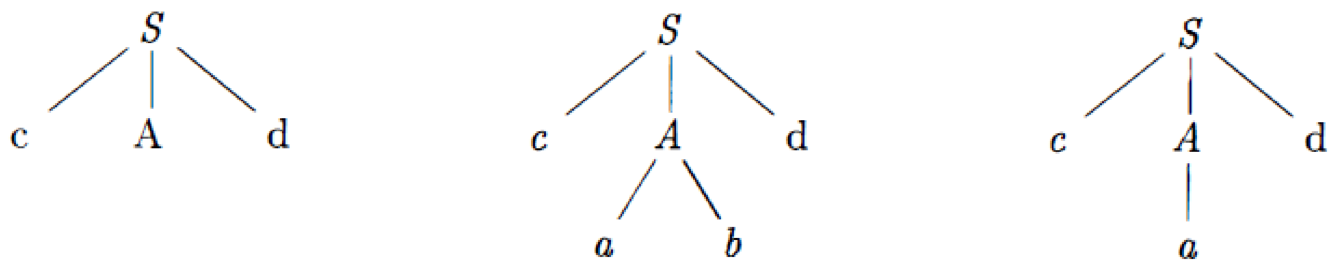
$S \rightarrow cAd$

$A \rightarrow ab \mid a$

$w = cad$

Après la lecture du premier symbole *c*, on se trouve avec l'arbre de la Figure (a). en avance le pointeur sur le symbole *a.* Le symbole non terminal le plus à gauche est *A*. nous remplaçons *A* avec la première alternative $A \rightarrow ab$ (Figure (b)), et nous avons une correspondance pour le deuxième symbole *c* de *w,* alors nous avançons le pointeur sur *d* qui ne correspond pas l'étiquète *b* de la feuil suivante. Dans ce cas nous reportons un échec et nous retournons à la deuxième alternative $A \rightarrow a$ (backtracking). Cette dernière ce termine avec succès (Figure (c)).

Lors du retour arrière, nous devons réinitialiser le pointeur d'entrée à la position 2.



(a) (b) (c)

Analyse descendante récursive

Consiste à écrire pour chaque symbole non-terminal A une procédure récursive A(). L'exécution commence par la procédure du symbole de départ, qui arrête et annonce succès si la procédure scanne toute la chaine en entrée.

```
void A() {
        Choose an A-production, A → X1 X2 ··· Xk;
        for ( i = 1 to k ) {
                if ( Xi is a nonterminal )
                        call procedure Xi();
                else if ( Xi equals the current input symbol a )
                        advance the input to the next symbol;
                else /* an error has occurred */;
        }
}
```

Le pseudo-code ci-dessus du symbole non-terminal *A* est non déterministe. Si $A \rightarrow A_1 \mid A_2 \mid \ldots \mid A_n$, et si au cours de l'analyse on a dérivé le symbole non-terminal *A*, alors le problème qui se pose est celui du choix de la règle à appliquer. Dans ce cas l'analyse récursive nécessite des retours arrière (backtracking) nécessitant des scannes répétés de la chaine en entrée pour chaque règle.

Pour permettre des retours arrière le pseudo-code doit être modifié. Comment ?

à la ligne (1) nous devons essayer chacune des alternatives de *A* dans un certain ordre. L'échec dans la ligne (7) n'est pas ultime, mais il faut retourner à la ligne (1) pour essayer une autre règle. Pour pouvoir essayer une autre règle, il faut réinitialiser le pointeur de la chaine en entrée sur la position dans laquelle il a été lors de ligne (1).

Analyse descendante prédicative (déterministe)

L'analyse descendante prédictive permet toujours de choisir une règle de production unique en se basant sur le prochain symbole de l'entrée et sans effectuer aucun retour en arrière. Elle s'applique à une classe restreinte de grammaires hots-contexte, dite grammaire **LL(k)**.

Il ya deux façons pour effectuer une analyse descendante prédictive :

- L'analyse prédictive récursive : implémentée avec des procédures récursives sans retour arrière
- L'analyse prédictive non récursive : dirigée par une table d'analyse

Ces deux types d'analyse nécessitent le calcule de deux ensembles :

- l'ensemble Premier (First), et
- l'ensemble Suivant (Follow).

➢Calcul de l'ensemble Premier

Soit $\alpha \in (N \cup T)^*$ une forme d'une ***GHC G = (N, T, P, S)***. On définit l'ensemble Premier(α) (ou First(α)), comme étant l'ensemble des terminaux qui peuvent apparaître au début d'une forme dérivable à partir de α . Formellement :

$$\Pr emier(\alpha) = \left\{ a \in T \,/\, \alpha \Rightarrow^* a\beta,\ \ \beta \in (N \cup T)^* \ \right\}$$

Algorithme de construction des ensembles Premier :

1- Si X est non-terminal et $X \rightarrow Y_1\, Y_2 \ldots.\, Y_n$ est une règle de la grammaire (Y_i symbole terminal ou non-terminal) alors :

1- ajouter les éléments de Premier(Y_1) sauf ε dans Premier(X),

2- s'il existe un j ($j \in \{\, 2,\ldots,n \,\}$) tel que pour tout i=1,…, j-1 on a $\varepsilon \in$ Premier (Y_i), alors ajouter les éléments de Premier(Y_j) sauf ε dans Premier(X)

3- si pour tout i=1,…,n on a $\varepsilon \in$ Premier (Y_i), alors ajouter ε dans Premier (X)

2- si $X \rightarrow \varepsilon$ est une règle de la grammaire, alors ajouter ε dans Premier (X)

3- Si X est terminal, alors Premier(X) = {X}

Recommencer jusqu'à ce qu'on n'ajoute rien de nouveau dans les ensembles Premier

Il est claire que

Premier(X) = Premier(X_1) $\cup$ Premier(X_2) $\cup$ … $\cup$ Premier(X_n), pour tout $X \rightarrow X_1 | X_2 | \ldots | X_n$

Exemple 1 :

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid -TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid /FT' \mid \varepsilon$
$F \rightarrow (E) \mid nb$

Premier(E) = Premier(T)= { (, nb }

Premier(E') = { + , - , ε }

Premier(T) =Premier(F) = { (, nb }

Premier(T') = { * , / , ε }

Premier(F) = { (, nb }

Exemple 2 :

$S \rightarrow ABCe$

$A \rightarrow aA \mid \varepsilon$

$B \rightarrow bB \mid cB \mid \varepsilon$

$C \rightarrow de \mid da \mid dA$

Premier(S) = Premier(A) $\cup$ Premier(B) $\cup$ Premier(C) = { a , b , c , d }

Premier(A) ={ a , ε }

Premier(B)={ b , c , ε }

Premier(C) = { d }

➢ Calcule de l'ensemble Suivant

Soit A un symbole non-terminal, Suivant(A) est l'ensemble de tous les symboles terminaux a qui peuvent apparaitre immédiatement à droite de A dans une dérivation $S \Rightarrow^{*} \alpha A a \beta$

$$Suivant(A) = \left\{ a \in T \,/\, S \Rightarrow^{*} \alpha A a \beta,\;\; avec\ \alpha, \beta \in (N \cup T)^{*} \right\}$$

Algorithme de construction des ensembles Suivant :

1- ajouter \$ dans Suivant(S), où S est l'axiome de la grammaire et \$ est le marqueur de fin de chaine d'entrée

2- pour chaque règle $A \rightarrow \alpha B \beta$ où B est un non-terminal, alors ajouter les éléments de Premier(β) dans Suivant(B) sauf ε

3- pour chaque règle $A \rightarrow \alpha B$, ajouter Suivant(A) à Suivant(B)

4- pou chaque règle $A \rightarrow \alpha B\beta$ avec $\varepsilon \in \Pr emeir(\beta)$, ajouter Suivant($A$) à Suivant($B$)

5- pou chaque règle $A \rightarrow Ba\beta$, avec a un symbole terminal, alors ajouter a à Suivant(B)

Exemple 1

$E \rightarrow TE'$
$E' \rightarrow +TE' \mid -TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid /FT' \mid \varepsilon$
$F \rightarrow (E) \mid nb$

Suivant(E) ={ $,) }

Suivant(E') = Suivant(E) = { $,) }

Suivant(T) = Premier(E') = { + , - ,) , $ }

Suivant(T') = Suivant(E') = { + , - ,) , $ }

Suivant(F)=Premier(T') ={ * , / , + , - , $,) }

Exemple 2 :

$S \rightarrow aSb \mid cd \mid SAe$

$A \rightarrow aAdB \mid \varepsilon$

$B \rightarrow bb$

	Premier	Suivant
S	{ a , c }	{ $, b , a , e }
A	{ a , ε }	{ d , e }
B	{ b }	{ d , e }

➢ Construction de la table d'analyse

La table d'analyse est un tableau M à deux dimensions qui indique pour chaque non-terminal A et chaque symbole terminal a ou le symbole \$ la règle de production à appliquer. La table d'analyse permet de choisir de façon explicite la production à appliquer lors de l'analyse.

Algorithme de construction :

- Pour chaque règle de production $A \to \alpha$ faire

  1- pour chaque $a \in \mathrm{Pr}\,emier(\alpha)$ (et $a \neq \varepsilon$), ajouter la production $A \to \alpha$ dans la case $M[A,a]$, 2- si $\varepsilon \in \mathrm{Pr}\,emier(\alpha)$, alors pour chaque $b \in Suivant(A)$ ajouter $A \rightarrow \alpha$ dans la case $M[A,b]$

- Chaque case $M[A,a]$ vide est une erreur syntaxique

Exemple

La table d'analyse de la grammaire de l'exemple 1

	nb	+	-	*	/	(	)	$
E	$E \rightarrow TE'$					$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$	$E' \rightarrow -TE'$				$E' \rightarrow \varepsilon$	$E' \rightarrow \varepsilon$
T	$T \rightarrow FT'$					$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$	$T' \rightarrow /FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow nb$					$F \rightarrow (E)$		

Grammaire LL(1)

On appelle grammaire **LL(1**) une grammaire pour la quelle chaque case de la table d'analyse contient au plus une règle. Dans le cas contraire, la grammaire n'est pas **LL(1)**.

LL(k) signifie :

- **L** : lecture de la chaine d'entrée de gauche à droite (scanning the input from **L**eft to right)
- **L** : création de l'arbre syntaxique par les dérivations les plus gauches (producing a **L**eftmost derivation).
- **k** : on lit **k** symbole de prévision à la fois du texte à analyser (with **k** symbols of lookahead).

On ne s'intéresse qu'aux grammaires **LL(1)**. Quand **k** > 1 les algorithmes de mise en œuvre deviennent difficiles.

Définition formelle d'une grammaire LL(1)

Une grammaire G est dite **LL(1)** si pour toute règle de production de la forme $A \rightarrow X_1 \mid X_2 \mid \ldots \mid X_n$ les conditions suivantes sont vérifiées:

$\forall i, j :$

1- Premier(X_i) $\cap$ Premier(X_j) $=$ $\varnothing$, pour tout $i \neq j$

2- S'il existe X_i annulable (c.-à-d. $X_i \rightarrow^* \varepsilon$), alors

Premier(X_j) $\cap$ Suivant(A) = $\varnothing$, pour tout $i \neq j$

3- Il existe au plus une règle $X_i \rightarrow^* \varepsilon$

Ces conditions entrainent l'unicité de l'existence d'une seule règle dans une case de la table d'analyse prédictive.

Avantages : cette technique est simple, très efficace et facile à implémenter.

Condition nécessaire pour qu'une grammaire soit LL(1)

Pour qu'une grammaire soit **LL(1)**, il faut qu'elle soit *non ambiguë*, *non récursive à gauche* et *factorisée à gauche*.

a- Récursivité à gauche

Une grammaire est récursive à gauche s'il existe un non-terminal *A* et une dérivation de la forme $A \to^{*} A\alpha$ o*ù* α est une chaine quelconque.

Cas particulier, on dit qu'on a une récursivité à gauche immédiate si la grammaire possède une production de la forme $A \to A\alpha$

- *Elimination de la récursivité à gauche immédiate*

  Remplacer toute règle de la forme $A \rightarrow A\alpha \mid \beta$ par

  $A \rightarrow \beta A'$

  $A' \rightarrow \alpha A' \mid \varepsilon$

  La grammaire obtenue après élimination de la récursivité à gauche génère le même langage que la grammaire initiale

Exemple 1

$S \rightarrow ScA \mid B$

$A \rightarrow Aa \mid \varepsilon$

$B \rightarrow Bb \mid d \mid e$

Cette grammaire contient plusieurs récursivités à gauche immédiates.

Apes élimination de la récursivité immédiate

$S \to BS'$

$S' \to cAS' \mid \varepsilon$

$A \to A'$

$A' \to aA' \mid \varepsilon$

$B \to dB' \mid eB'$

$B' \to bB' \mid \varepsilon$

De façon générale **la récursivité à gauche immédiate** peut être éliminée comme suite :

1- Grouper toutes les *A*-productions (les règle de productions ayant la même partie droite *A*): $A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

où aucun β_i ne commence par A

2- Remplacer les A-productions par :

$$A \to \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$$

$$A' \to \alpha_1 A' \mid \alpha 2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$$

- *Elimination de la récursivité à gauche*

Exemple 2

$$S \to Aa \mid b$$

$$A \to Ac \mid Sd \mid \varepsilon$$

Le non terminal S est récursive à gauche car $S \to Aa \to Sda$ mais pas immédiatement récursive à gauche.

L'algorithme ci-dessous, systématiquement élimine les récursivité à gauche d'une grammaire ne contenant aucun cycle (c.-à-d. des dérivations de la forme $A \Rightarrow^{*} A$) ou des ε -règles (c.-à-d. règle de la forme $A \rightarrow \varepsilon$).

Début

Ordonner les non-terminaux $A_1, A_2, \ldots, A_n$

pour i=1 à n faire

pour j=1 à (i-1) **faire**

remplacer chaque production de la forme $A_i \rightarrow A_j\alpha$ où

$A_j \rightarrow \beta_1 \mid \beta_2 \mid \ldots \mid \beta_p$ par $A_i \rightarrow \beta_1\alpha \mid \beta_2\alpha \mid \ldots \mid \beta_p\alpha$

fin pour

éliminer les récursivités à gauche immédiates des productions A_i

fin pour

Fin

Exemple 2

On ordonne *S, A*

$i = 1$ pas de récursivité immédiate dans $S \to Aa \mid b$

$i = 2$ et j=1 on obtient $A \to Ac \mid Aad \mid bd \mid \varepsilon$

on élimine la récursivité immédiate :

$$A \to bdA' \mid A'$$
$$A' \to cA' \mid adA' \mid \varepsilon$$

On obtient la grammaire

$$S \to Aa \mid b$$
$$A \to bdA' \mid A'$$
$$A' \to cA' \mid adA' \mid \varepsilon$$

b- Factorisation à gauche

Nous cherchons à écrire des analyseurs prédictifs. Cela veut dire qu'à tout moment le choix entre productions qui ont le même membre gauche doit pouvoir se faire, sans risque d'erreur, en comparant le symbole courant de la chaine à analyser avec les symboles susceptibles de commencer les dérivations des membres droits des productions en compétition.

Une grammaire contenant des productions comme $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$ viole ce principe.

La factorisation à gauche corrige ce problème comme suit :

Pour chaque non-terminal A :

- trouver le préfixe le plus long α ($\alpha \neq \varepsilon$) commun à deux ou plusieurs de ses alternatives.
- Remplacer $A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \ldots \mid \alpha\beta_n \mid \lambda_1 \mid \ldots \mid \lambda_p$ (où λ_i ne commencent pas par α) par

$$A \to \alpha A' \mid \lambda_1 \mid \ldots \mid \lambda_p$$

$$A' \to \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

Répéter pour tout non-terminal ayant un préfixe commun à ses alternatives.

Exemple

$stmt \rightarrow$ **if** $expr$ **then** $stmt$ **else** $stmt$ | **if** $expr$ **then** $stmt$

$$stmt \rightarrow \underbrace{\textbf{if}\ \ expr\ \ \textbf{then}\ \ stmt}_{\alpha}\underbrace{\textbf{else}s\ \ stmt}_{\beta} \mid \underbrace{\textbf{if}\ \ expr\ \ \textbf{then}\ \ stmt}_{\alpha}$$

$stmt \rightarrow$ **if** $expr$ **then** $stmt\ A'$

$A' \rightarrow$ **else** $stmt$ | ε

Cette grammaire n'est pas LL(1)

	Premier	Suivant
Stmt	{if}	{$,else}
A'	{else, ε}	{$,else}

	If	else	$
Stmt	*stmt* → **if** *expr* **then** *stmt A'*		*A'* → ε
A'		*A'* → **else** *stmt* *A'* → ε	

Avec la grammaire factorisée, pour la chaine d'entrée : **if** *a* **then if** *b* **then** *c* **elese** *d* il ya deux arbre de dérivation. La grammaire factorisée est **ambigüe.**

Règle pour levé l'ambigüité :

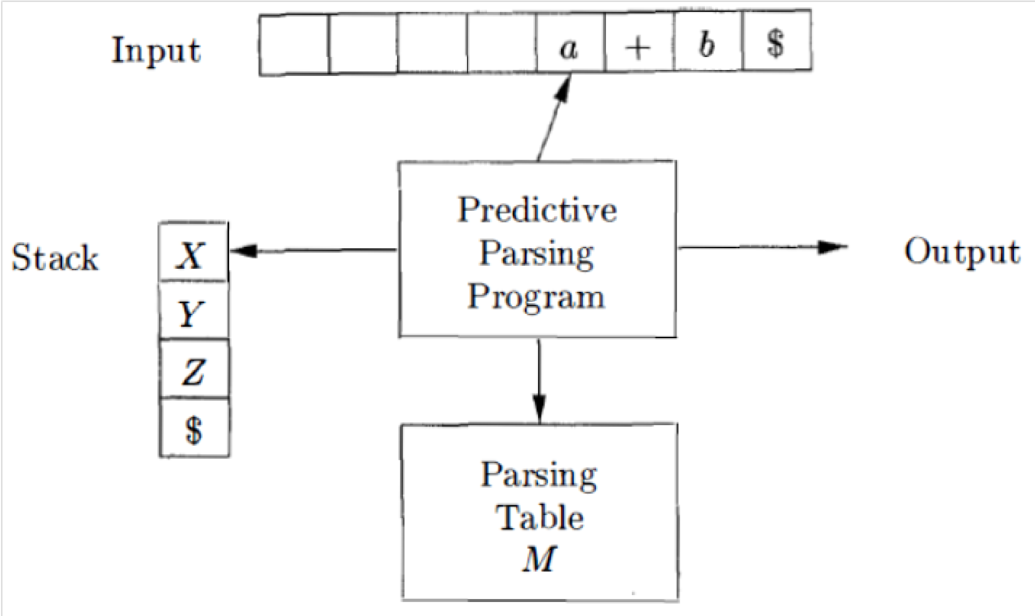
Dans la dérivation d'un non-terminal, une "ε-règle ne peut être choisie que lorsque aucune autre production n'est applicable.

Dans cet exemple, cela donne : si la chaine d'entrée commence par **else** alors on doit nécessairement choisir la première règle.

Algorithme d'analyseur LL(1) dirigé par la table d'analyse

Comme le montre la figure, un analyseur syntaxique prédictif dirigé par une table d'analyse est composé des éléments suivants :

- Un **tampon d'entrée (Input)**: contient la chaine à analyser, suivie du symbole $.
- Une **pile (Stack)**: contient une séquence de symboles (terminaux ou non-terminaux), avec un symbole $ qui marque le **fond de la pile**. Au départ, la pile contient l'**axiome** de la grammaire au sommet et le symbole $ au **fond**.
- Une **table d'analyse LL(1) (Parsing Table)**.
- Un **flot de sortie (Output) (l'arbre syntaxique)**



Algorithme

Entrée : une chaine *w* terminée par $, *ps* est le pointeur sur le premier symbole de *w* et la table d'analyse M de la grammaire

Sorite : si *w* est dans *L(G)*, arbre d'analyse de *w*, sinon erreur

Répéter
X : reçoit le symbole en sommet de pile ; a : est le symbole pointé par *ps*
Si (X est un non-terminal) **alors**
    Si ($M[X,a]$ = $X \rightarrow Y_1Y_2...Y_n$) **alors**
        dépiler X ; empiler Y_n suivie Y_{n-1} jusqu'à Y_1 ; emmètre la règle $X \rightarrow Y_1Y_2...Y_n$ en sortie
    Sinon (*case vide dans la table d'analyse***)**
        ERREUR
    finsi
Sinon
    Si X= $ **alors**
        Si a = $ **alors ACEPTER**
        Sion ERREUR
        finsi
    Sinon
        Si ***X* = a alors**   dépiler ***X* ;** avancer *ps*
        Sinon ERREUR
        finsi
    finsi
finsi
Jusqu'à ACCEPTER ou **ERREUR**

Exemple

Soit la grammaire G

$E \rightarrow E+T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

La grammaire G' équivalant à G après élimination des récursivités à gauche

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid nb$

- *Calcul des ensembles Premier et Suivant :*

	Premier	**Suivant**
E	{ (, nb }	{) , $ }
E'	{ + }	{) , $ }
T	{ (, nb }	{ + ,) , $ }
T'	{ * }	{ + ,) , $ }
F	{ (, nb }	{ + , * ,) , $ }

- *Construction de la table d'analyse*

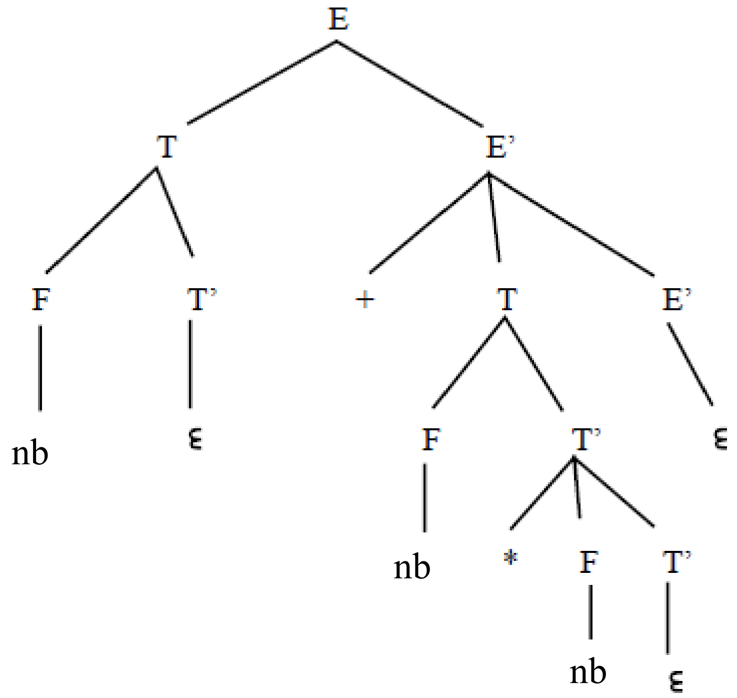
	+	*	(	)	nb	$
E			$E \rightarrow TE'$		$E \rightarrow TE'$	
E'	$E' \rightarrow +TE'$			$E' \rightarrow \varepsilon$		$E' \rightarrow \varepsilon$
T			$T \rightarrow FT'$		$T \rightarrow FT'$	
T'	$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$		$T' \rightarrow \varepsilon$
F			$F \rightarrow (E)$		$F \rightarrow nb$	

Analysons la chaine ***w* = nb+nb*nb**

Pile	Entrée	Sortie
$E	**nb+nb*nb$**	E->TE’
$E’T	**nb+nb*nb$**	T->FT’
$E’T’F	**nb+nb*nb$**	F->nb
$E’T’nb	**nb+nb*nb$**	
$E’T’	**+nb*nb$**	T’->ε
$E’	**+nb*nb$**	E’->+TE’
$E’T+	**+nb*nb$**	
$E’T	**nb*nb$**	T->FT’
$E’T’F	**nb*nb$**	F->nb
$E’T’nb	**nb*nb$**	
$E’T’	***nb$**	T’->*FT’

$E'T'F*	***nb$**	
$E'T'F	**nb$**	F->nb
$E'T'nb	**nb$**	
$E'T'	**$**	T'->ε
$E'	**$**	E'->ε
$	**$**	**ACCEPTE**

L'arbre syntaxique de la chaine ***w*** **= nb+nb*nb**



Analyse Ascendante