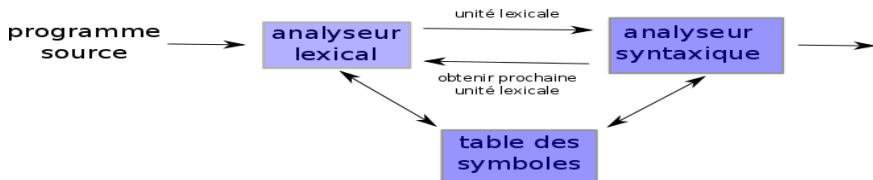


Analyse syntaxique



- ▶ L'analyseur lexical a en charge la reconnaissance des unités lexicales (nombres, mots clés, identifiants),
- ▶ L'analyseur syntaxique prend en charge l'agencement des unités lexicales pour former un langage.
- ▶ Les deux analyseurs collaborent via la table des symboles.

Exemple :

Si $(x < 0)$

alors

$x = -x$

Finsi

→

Si Expr

alors

Instr

Finsi

Analyse Syntaxique et grammaires

Nous allons utiliser les grammaires car :

- ▶ Les grammaires permettent une spécification facile et précise d'un langage,
- ▶ Pour les classes de grammaires usuellement utilisées, il existe des outils efficaces (Yacc, Bisons,. . .) permettant d'analyser automatiquement un fichier source à partir d'une grammaire.
- ▶ Les grammaires aident à spécifier proprement les langages de programmation ce qui est utile dans toutes les étapes ultérieures.
- ▶ Une spécification d'un langage par une grammaire permet de le faire évoluer (ajouter des fonctionnalités) simplement.

Grammaire non contextuelle

Une grammaire non contextuelle (/hors contexte/algébrique) est composée de productions :

$$X \rightarrow w$$

où X est appelé un non terminal (membre gauche de la règle) et w est une chaîne composée de terminaux et/ou de non terminaux.

Le terme non contextuel vient du fait que l'on remplace X par w sans référence au contexte où il apparaît.

Exemple de grammaire :

$$expr \rightarrow expr \ op \ expr$$

$$expr \rightarrow (\ expr \)$$

$$expr \rightarrow - \ expr$$

$$expr \rightarrow Id$$

$$op \rightarrow +$$

$$op \rightarrow -$$

$$op \rightarrow *$$

$$op \rightarrow /$$

► $expr$ et op sont des non terminaux

► id , $+$, $-$, $*$, $/$ sont des terminaux

► $expr$ représente l'axiome : tout le langage est dérivé de l'axiome.

Notations

1. Terminaux : Les lettres minuscules du début de l'alphabet : a, b, c. . . , les symboles d'opérateurs +, -, . . . , les symboles de ponctuation '(', ')', ',', ';', '...', les chiffres 0 à 9 les chaînes en gras **id**, **Si** représentant des unités lexicales.
2. Non terminaux : Les lettres majuscules du début de l'alphabet : A, B, C. . . , la lettre S qui représente généralement l'axiome, les noms en italiques *expr*, *op*
3. Les lettres majuscules de la fin de l'alphabet X, Y, Z représentent des symboles grammaticaux (i.e. des terminaux ou non terminaux)
4. les chaînes minuscules de la fin de l'alphabet u, v, w, . . . , z représentent des chaînes de terminaux (ou phrases).
5. Les lettres grecques minuscules α , β , γ représentent des chaînes de symboles grammaticaux ou proto-phrases (composées de terminaux et non terminaux).
Ex : $A \rightarrow \alpha$

Notations

6. La suite de productions $A \rightarrow \alpha_1, A \rightarrow \alpha_2, \dots, A \rightarrow \alpha_n$ se réécrit en $A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$.
7. Sauf indication contraire la partie gauche de la première production indique l'axiome.
 - La notation $E \Rightarrow -E$ signifie E se dérive en $-E$.
Exemple : $E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(Id)$
De façon plus générale $\alpha A \beta \Rightarrow \alpha \gamma \beta$ si $A \rightarrow \gamma$.
 - La notion se dérive en 0 à n étapes se note $\Rightarrow^*$. On a ainsi $E \Rightarrow^* -(Id)$
mais également $E \Rightarrow^* -(E)$.
 - $\Rightarrow^+$ signifie se dérive en 1 à n étapes.

Vérification et expressions régulières

Le langage engendré par une grammaire est l'ensemble des phrases dérivées de son axiome. On dit dans ce cas que le langage est non contextuel.

$$L(G) = \{ w \in \Sigma^* \mid S \xRightarrow{*} w \}$$

- La preuve qu'une grammaire engendre un langage particulier se fait rarement sur des grammaires complètes mais peut se faire (et se fait) sur des parties de celles-ci ou des grammaires simples. On montre que :
 $\forall w \in \Sigma^*$ tel que $S \xRightarrow{*} w$ on a $w \in L(G)$ et inversement, pour tout $w \in L(G)$ il existe une séquence de réduction telle que $S \Rightarrow w$.
- Toute expression régulière peut être codée par une grammaire non contextuelle. Le contraire n'est évidemment pas vrai. On restreint l'analyse lexicale à la reconnaissance des unités lexicales.

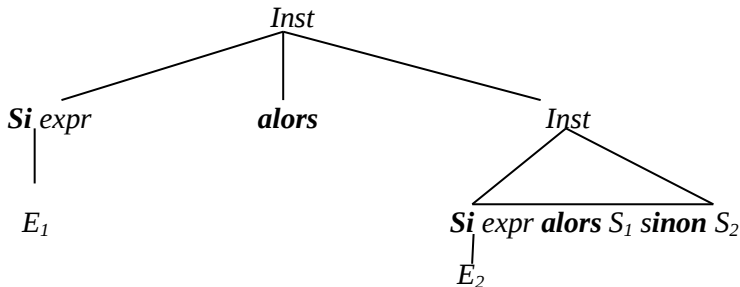
Suppression des ambiguïtés

La grammaire suivante :

$$\begin{array}{lcl} inst & \rightarrow & \text{si } expr \text{ alors } inst \\ & & | \text{ si } expr \text{ alors } inst \text{ sinon } inst \\ & & | \text{ autre} \end{array}$$

produit une ambiguïté sur la proto-phrase :

si E_1 alors si E_2 alors S_1 sinon S_2



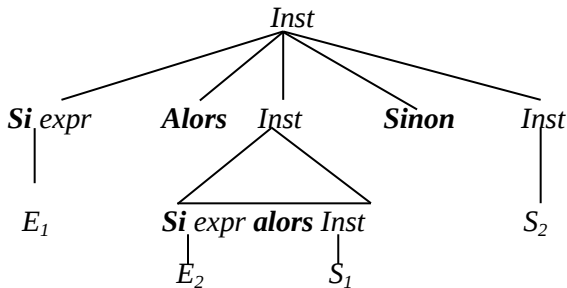
Suppression des ambiguïtés

La grammaire suivante :

$$\begin{array}{lcl} inst & \rightarrow & \text{si } expr \text{ alors } inst \\ & & | \text{ si } expr \text{ alors } inst \text{ sinon } inst \\ & & | \text{ autre} \end{array}$$

produit une ambiguïté sur la proto-phrase :

si E_1 alors si E_2 alors S_1 sinon S_2



Suppression des ambiguïtés

Solution : Affecter chaque sinon au alors précédant le plus proche. Réécrire la grammaire de la façon suivante.

$inst$	$\rightarrow$	$instFermee$ $ instNonFerme$
$instFermee$	$\rightarrow$	$si\ expr\ alors\ instFermee\ sinon\ instFermee$ $ autre$
$instNonFermee$	$\rightarrow$	$si\ expr\ alors\ inst$ $ si\ expr\ alors\ instFermee\ sinon\ instNonFermee$

On interdit les constructions du type :

$si\ E_1\ alors\ (si\ E_2\ alors\ S_1)\ sinon\ S_2$

qui seront interprétées comme :

$si\ E_1\ alors\ (si\ E_2\ alors\ S_1\ sinon\ S_2)$

Suppression de la récursivité à gauche

Récursivité directe

Une grammaire récursive à gauche comporte des règles telles que $A \xRightarrow{+} A\alpha$. Ce genre de dérivation ne peut pas être traité par des méthodes d'analyse descendante (celles que l'on étudie). On a donc besoin de transformer les règles $A \rightarrow A\alpha|\beta$ en les productions équivalentes:

$$\begin{aligned} A &\rightarrow \beta A' \\ A' &\rightarrow \alpha A' | \epsilon \end{aligned}$$

Plus généralement si $A \rightarrow A\alpha_1|\dots|A\alpha_n|\beta_1|\dots|\beta_m$, où chaque β_i ne commence pas par A . On remplace cette production par :

$$\begin{aligned} A &\rightarrow \beta_1 A' | \beta_2 A' | \dots | \beta_m A' \\ A' &\rightarrow \alpha_1 A' | \alpha_2 A' | \dots | \alpha_n A' | \epsilon \end{aligned}$$

Suppression de la récursivité à gauche

Exemple :

$$\begin{aligned}E &\rightarrow E + T \mid T \\T &\rightarrow T * F \mid F \\F &\rightarrow (E) \mid \mathbf{id}\end{aligned}$$

se réécrit en :

$$\begin{aligned}E &\rightarrow TE' \\E' &\rightarrow +TE' \mid \epsilon \\T &\rightarrow FT' \\T' &\rightarrow *FT' \mid \epsilon \\F &\rightarrow (E) \mid \mathbf{id}\end{aligned}$$

Suppression de la récursivité à gauche: récursivité indirecte

La récursivité à gauche peut être indirecte par exemple :

$$S \rightarrow Aa|b$$

$$A \rightarrow Ac|Sd|\epsilon$$

On a la dérivation $S \xRightarrow{+} Sda$, ($S \Rightarrow Aa \Rightarrow Sda$).

- ® On ordonne les non terminaux et on supprime itérativement les réductions permettant $A_i \Rightarrow A_j \alpha \Rightarrow A_i \beta \alpha$. Puis on supprime les récursivités gauches.

procédure suprRecGauche (Grammaire S)

Déclaration i, j : Entiers

début Ordonner les non terminaux $A_1, \dots, A_n$

pour i $\leftarrow 1$ à n faire

pour j $\leftarrow 1$ à i-1 faire

Remplacer $A_i \rightarrow A_j \gamma$ par $A_i \rightarrow \delta_1 \gamma | \delta_2 \gamma | \dots | \delta_k \gamma$

Où $A_j \rightarrow \delta_1 | \delta_2 | \dots | \delta_k$ sont les A_j productions courantes

finpour

supprimer les récursivités gauches immédiates des A_i productions

finpour

fin

Suppression de la récursivité à gauche : récursivité indirecte

Exemple: Soit la grammaire suivante avec l'ordre S, A .

$$\begin{aligned} S &\rightarrow Aa|b \\ A &\rightarrow Ac|Sd|\epsilon \end{aligned}$$

La première itération ne modifie pas $S \rightarrow Aa | b$. La seconde remplace $A \rightarrow Sd$ par :

$$A \rightarrow Aad|bd$$

qui s'ajoutent aux productions $A \rightarrow Ac|\epsilon$. On a donc $A \rightarrow Aad|Ac|bd|\epsilon$ dont on supprime la récursivité gauche immédiate pour obtenir :

$$\begin{aligned} S &\rightarrow Aa|b \\ A &\rightarrow bdA'|A' \\ A' &\rightarrow cA'|adA'|\epsilon \end{aligned}$$

Factorisation à gauche

Différentes productions peuvent avoir les mêmes préfixes. Par exemple :

$$\begin{array}{lcl} inst & \rightarrow & \text{si } expr \text{ alors } inst \\ & & | \text{ si } expr \text{ alors } inst \text{ sinon } inst \\ & & | \text{ autre} \end{array}$$

Sur rencontre de *si expr alors inst* laquelle des deux règles appliquer ? On explicite ce choix par une factorisation gauche :

$$\begin{array}{lcl} inst & \rightarrow & \text{si } expr \text{ alors } inst \ S' \\ S' & \rightarrow & \text{sinon } instr \ | \ \epsilon \end{array}$$

Cette dernière grammaire est plus efficace car le choix ne s'effectue que quand il doit se faire.

Factorisation à gauche

De façon plus générale si il existe $\alpha \neq \varepsilon$ tel que:

$$A \rightarrow \alpha\beta_1|\alpha\beta_2|\dots|\alpha\beta_n|\gamma$$

on remplace ces productions par:

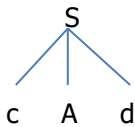
$$A \rightarrow \alpha A'|\gamma$$

$$A' \rightarrow \beta_1|\beta_2|\dots|\beta_n$$

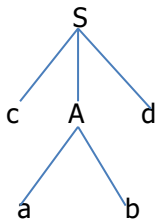
Le problème du rebroussement

Soit la grammaire :

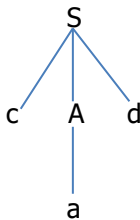
$$\begin{aligned} S &\rightarrow cAd \\ A &\rightarrow ab|a \end{aligned}$$



(a)



(b)



(c)

et la chaîne $w = cad$. On développe $S(a)$. Le pointeur d'entrée est positionné sur c . La feuille la plus à gauche étant égale à c nous avançons le pointeur d'entrée sur a (second symbole de w) et appliquons la première production de A (b). On lit a sur le tampon d'entrée et la feuille la plus à gauche, on

Le problème du rebroussement

avance le pointeur d'entrée sur d et lisons la feuille b : Erreur. On backtraque donc et ramenons le pointeur sur c , on applique ensuite la deuxième production de A qui nous permet de reconnaître w .

- On comprend de l'exemple précédent que des grammaires récursives à gauche peuvent faire boucler des analyseurs prédictifs.
- Notre objectif va être de supprimer autant que possible le backtracking quitte à restreindre les grammaires.

Grammaires et diagrammes de transitions

On associe un diagramme à chaque non terminal. Pour chaque production $A \rightarrow X_1 \dots X_n$ on crée un chemin de l'état initial à l'état final avec les transitions $X_1 \dots X_n$.

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' | \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' | \epsilon \\ F &\rightarrow (E) | \text{id} \end{aligned}$$

Exemple sur la grammaire d'expressions régulières simplifiées.

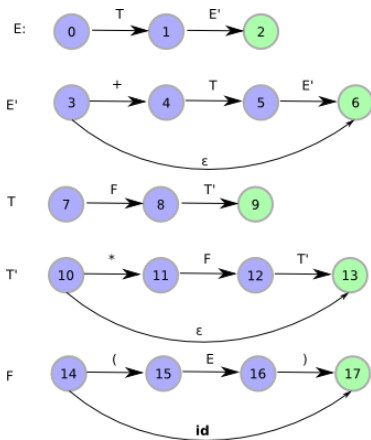
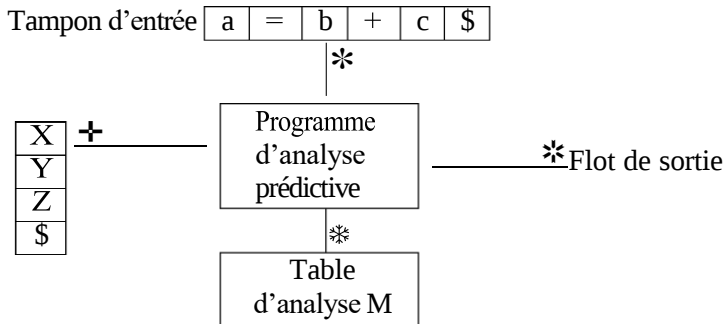


Schéma d'un analyseur syntaxique prédictif

- ▶ Commencer dans l'état de départ de l'axiome.
- ▶ Si on est dans un état s avec une transition (s, t) étiquetée par le terminal a , et si le prochain symbole d'entrée est a aller en t et décaler l'entrée d'un cran sur la droite.
- ▶ Si (s, t) est étiquetée par un non terminal A , aller dans l'état de départ de A . Si on atteint l'état final de A , passer à t .
- ▶ si (s, t) est étiquetée par ϵ on passe directement à t .

On doit donc lire le tampon d'entrée et empiler au moins les règles qu'on utilise.

Analyse prédictive



L'analyseur lit le tampon d'entrée et décide de la prochaine action à effectuer en fonction du sommet de pile et de la table M .

Analyse prédictive

Plus précisément: Si X est le sommet de pile et a l'entrée courante.

- ▶ Si $X = a = \$$ l'analyseur s'arrête et indique succès.
- ▶ Si $X = a \neq \$$ l'analyseur dépile X et décale le tampon d'entrée.
- ▶ Si X est un non terminal. L'analyseur consulte $M[X, a]$.
- ▶ Si $M[X, a] = \text{erreur}$, l'analyseur indique une erreur.

Sinon $M[X, a]$ est une production $X \rightarrow UVW$ et l'analyseur dépile X et empile W , V et U (dans cet ordre)

Analyse prédictive : Algorithme

procédure AnalysePredictive (P : pile, M : table, T : tampon)

Déclaration ps : pointeur sur T

début

répéter

X ← sommet de P

a ← valeur pointée par ps

si X est un terminal ou \$ alors

si X=a alors dépiler X, avancer ps sinon Erreur()

sinon

si $M[X, a] = X \rightarrow Y_1 \dots Y_n$ alors

dépiler X, Empiler $Y_n, \dots, Y_1$

Emmètre en sortie $X \rightarrow Y_1 \dots Y_n$

sinon

Erreur()

finsi

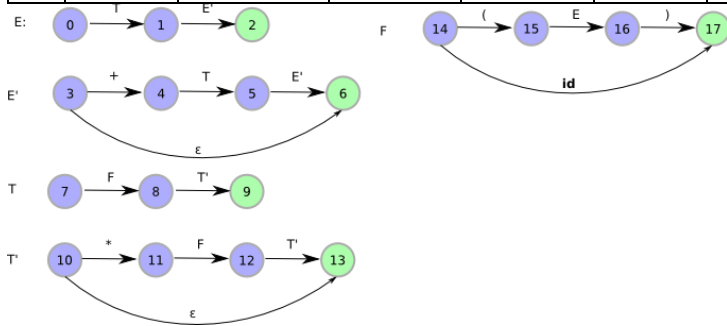
finsi

jusqu'à ce que $X = \$$

fin

Exemple de table d'analyse

	id	+	*	(	)	\$
E	$E \rightarrow TE'$			$E \rightarrow TE'$		
E'		$E' \rightarrow +TE'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$		



FIRST et FOLLOW

Une question récurrente en compilation est: L'application de cette production me permettra t-elle de décaler tel symbole sur le tampon d'entrée ?. Les fonctions FIRST et FOLLOW nous aident à y répondre.

- ▶ Si α est une proto-phrase, $FIRST(\alpha)$ est l'ensemble des terminaux a tel que $\alpha \Rightarrow^* aw$. Si $\alpha \Rightarrow^* \epsilon$ alors $\epsilon \in FIRST(\alpha)$.
- ▶ Si A est un non terminal $FOLLOW(A)$ est l'ensemble des terminaux a tel qu'il existe une dérivation $S \Rightarrow^* \alpha A a \beta$. Si A est le symbole le plus à droite d'une proto-phrase ($S \Rightarrow^* \alpha A$) alors $\$ \in FOLLOW(A)$.

Calcul de FIRST

Calcul de $FIRST(X)$, pour tout symbole X de la grammaire

1. Si X est un terminal, $FIRST(X) = \{X\}$
2. Si $X \rightarrow \epsilon$, ajouter ϵ à $FIRST(X)$
3. Si X est un non terminal et $X \rightarrow Y_1 \dots Y_k$.

procédure UpdateFirst ($FIRST(X)$, $Y_1, \dots Y_k$)

début

$i \leftarrow 1$

répéter

$FIRST(X) \leftarrow FIRST(X) \cup FIRST(Y_i)$

$i \rightarrow i+1$

jusqu'à ce que $\epsilon \notin FIRST(Y_{i-1})$

fin

Le calcul de $FIRST(X_1 \dots X_n)$ se fait par une méthode analogue à UpdateFirst.

Calcul de FOLLOW

1. Ajouter \$ à FOLLOW(S), S l'axiome, \$ la fin de chaîne.

2. Pour toute production $B \rightarrow \alpha A \beta$,

$$FOLLOW(A) \leftarrow FOLLOW(A) \cup (FIRST(\beta) - \{\epsilon\}).$$

3. Pour toute production $B \rightarrow \alpha A$ ou $B \rightarrow \alpha A \beta$ avec $\epsilon \in FIRST(\beta)$,

$$FOLLOW(B) \leftarrow FOLLOW(A) \cup FOLLOW(B).$$

Calcul de FIRST et FOLLOW: Exemple

Reprenons notre grammaire:

$$\begin{aligned}E &\rightarrow TE' \\ E' &\rightarrow +TE' | \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' | \epsilon \\ F &\rightarrow (E) | \text{id}\end{aligned}$$

$$\begin{aligned}\text{FIRST}(F) &= \{ (, \text{id} \} \\ \text{FIRST}(T') &= \{ *, \epsilon \} \\ \text{FIRST}(T) &= \{ (, \text{id} \} \\ \text{FIRST}(E') &= \{ +, \epsilon \} \\ \text{FIRST}(E) &= \{ (, \text{id} \}\end{aligned}$$

$$\begin{aligned}\text{FOLLOW}(E) &= \{ \$,) \} \\ \text{FOLLOW}(E') &= \{ \$,) \} \\ \text{FOLLOW}(T) &= \{ +, \$,) \} \\ \text{FOLLOW}(T') &= \{ +, \$,) \} \\ \text{FOLLOW}(F) &= \{ +, \$,), * \}\end{aligned}$$

Construction des tables

- ▶ La production $A \rightarrow \alpha$, peut être utilisée si l'entrée sur le tampon d'entrée appartient à $FIRST(\alpha)$.
- ▶ Si $\alpha \xRightarrow{*} \epsilon$, on peut toujours utiliser cette règle si le symbole d'entrée appartient à $FOLLOW(A)$ (y compris si ce symbole est \$).

Donc on construit la table de la façon suivante:

1. Pour chaque production $A \rightarrow \alpha$ faire les étapes 2 et 3.
2. Pour chaque terminal a dans $FIRST(\alpha)$, ajouter $A \rightarrow \alpha$ à $M[A, a]$
3. Si $\epsilon \in FIRST(\alpha)$, ajouter $A \rightarrow \alpha$ à $M[A, b]$ pour tout $b \in FOLLOW(A)$. Si $\epsilon \in FIRST(\alpha)$ et $\$ \in FOLLOW(A)$ ajouter $A \rightarrow \alpha$ à $M[A, \$]$.
4. Toute entrée de M non remplie correspond à une erreur.

Analyse Prédicative: Exemple

Reprenons la grammaire G précédente et la table d'analyse correspondante. L'algorithme de l'analyse prédictive non récursive appliqué à G et à $w = \text{id+id*id}$ donne la séquence suivante :

PILE	TAMPON	SORTIE
\$E	id+id*id\$	
\$E'T	id+id*id\$	$E \rightarrow TE'$
\$E'T'F	id+id*id\$	$T \rightarrow FT'$
\$E'T'id	id+id*id\$	$F \rightarrow \text{id}$
\$E'T'	+id*id\$	
\$E'	+id*id\$	$T \rightarrow \epsilon$

Analyse Prédictive: Exemple

$\$E'T+$	$+id*id\$$	$E' \rightarrow +TE'$
$\$E'T$	$id*id\$$	
$\$E'T'F$	$id*id\$$	$T \rightarrow FT'$
$\$E'T'id$	$id*id\$$	$F \rightarrow \mathbf{id}$
$\$E'T'$	$*id\$$	
$\$E'T'F*$	$*id\$$	$T' \rightarrow *FT'$
$\$E'T'F$	$id\$$	
$\$E'T'id$	$id\$$	$F \rightarrow \mathbf{id}$
$\$E'T'$	$\$$	
$\$E'$	$\$$	$T' \rightarrow \epsilon$
$\$$	$\$$	$E' \rightarrow \epsilon$

Grammaires LL(1)

Toute grammaire dont la table remplie par l'algorithme précédent ne comporte pas d'entrée multiple est appelée LL(1): Left to right scanning, Leftmost derivation, utilisant un symbole de prévision.

Une grammaire est LL(1) ssi pour toute production $A \rightarrow \alpha | \beta$

1. Pour aucun terminal a nous avons $\overset{*}{\alpha} \Rightarrow aw$ et $\overset{*}{\beta} \Rightarrow aw$,
2. Au plus une des chaînes α ou β peut se dériver en la chaîne vide,
3. Si $\overset{*}{\beta} \Rightarrow \epsilon$, α ne se dérive pas en une chaîne commençant par un terminal de FOLLOW(A).

Notre grammaire des expressions arithmétiques est LL(1). La grammaire correspondant au si sinon ne l'est pas.

Grammaires LL(1) : Conclusions

- ▶ Les grammaires LL(1) restent suffisamment simples pour pouvoir être développées 'à la main'.
- ▶ Le pouvoir expressif de ces grammaires reste très limité.