# Analyse syntaxique descendante

F. Alexandre

École Supérieure d'Informatique et Applications de Lorraine

17 mars 2009

# Plan et objectifs

- Initiation à l'analyse syntaxique
- Problèmes : étant donnée une grammaire algébrique
  - déterminer si un mot appartient au langage défini par la grammaire
  - construire l'arbre syntaxique d'un mot appartenant au langage défini par la grammaire
- Plan
  - Rappels sur les grammaires algébriques
  - Descente récursive
  - Définition des premiers et suivants

## Rappels : grammaire algébrique

**Definition (Grammaire algébrique)**

Une grammaire algébrique est un quadruplet $G = (N, T, \rightarrow, S)$ tel que

- $N$ est l'ensemble des symboles non terminaux.
- $T$ est l'ensemble des symboles terminaux.
- $\rightarrow$ est une relation de $N$ vers $(N \cup T)^*$ telle que pour chaque élément $A$ de $N$ il n'y ait qu'un nombre fini de mots $\alpha$ appartenant à $(N \cup T)^*$ tels que $A \rightarrow \alpha$.
- $S$ est un élément de $N$, appelé l'axiome de la grammaire.

**Definition (Réécriture)**

La relation de réécriture $\rightarrowtail$ est définie sur $(N \cup T)^*$. $\alpha$ se réécrit en $\beta$, qu'on note $\alpha \rightarrowtail \beta$, si $\alpha = \alpha_1 A \alpha_2$ et $\beta = \alpha_1 \gamma \alpha_2$ où

$\alpha_1,\ \alpha_2,\ \gamma \in (N \cup T)^*$, $A \in N$ et $A \rightarrow \gamma$ est une règle de $G$.

**Definition (Dérivation non stricte et stricte)**

La relation de dérivation est la fermeture réflexive transitive de la relation $\rightarrowtail$, on la note $\overset{*}{\rightarrowtail}$.

$\alpha$ se dérive en $\beta$ que l'on note $\alpha \overset{*}{\rightarrowtail} \beta$, si on passe de $\alpha$ à $\beta$ par une suite finie de réécriture.

## Réécriture gauche, dérivation gauche

**Definition (Réécriture à gauche)**

La relation de réécriture à gauche $\overset{g}{\rightarrowtail}$ est définie sur $(N \cup T)^*$. $\alpha$ se réécrit à gauche en $\beta$, qu'on note $\alpha \overset{g}{\rightarrowtail} \beta$, si $\alpha = wA\alpha_1$ et $\beta = w\gamma\alpha_1$ où

$\alpha_1,\ \gamma \in (N \cup T)^*$, $w \in T^*$, $A \in N$ et $A \rightarrow \gamma$ est une règle de $G$.

**Remarque** : la réécriture à gauche consiste à réécrire le non terminal le plus à gauche.

**Definition (Dérivation à gauche)**

La relation de dérivation à gauche est la fermeture réflexive transitive de la relation de réécriture à gauche.

## Arbre syntaxique

### Definition

Soit $G = (N,\ T,\ \rightarrow,\ S)$, un arbre syntaxique pour la grammaire $G$ est un arbre étiqueté par les éléments de $N \cup T \cup \{\varepsilon\}$ qui satisfait les conditions suivantes :

- la racine de l'arbre est étiquetée par $S$, l'axiome de $G$
- chaque nœud interne est étiqueté par un élément de $N$. Chaque feuille est étiquetée par un élément de $T$ ou par $\varepsilon$.
- pour tout nœud interne, si son étiquette est un non-terminal $A$ et si ses descendants immédiats sont les nœuds $n_1, \ldots, n_k$ ayant respectivement pour étiquettes $X_1, \ldots, X_k$ alors $A \rightarrow X_1 \ldots X_k$ doit être une règle de production de $G$.
- si un nœud est étiqueté par $\varepsilon$, alors ce nœud est le seul descendant immédiat de son prédécesseur (cette dernière règle évite l'introduction d'instances inutiles de $\varepsilon$ dans l'arbre syntaxique).

### Definition (Mot généré par un arbre syntaxique)

Un mot généré par un arbre syntaxique est obtenu par la concaténation des feuilles de l'arbre prises de gauche à droite.

## Langage engendré par une grammaire algébrique

### Definition

Soit $G = (N,\ T,\ \rightarrow,\ X)$ une grammaire algébrique, le langage $L(G)$ engendré par $G$ est

1. $L(G) = \{w; w \in T^* \text{ et } X \overset{*}{\rightarrowtail} w\}$
2. $L(G) = \{w; w \in T^* \text{ et } w \text{ est un mot généré par un arbre syntaxique}\}$

### Definition (Grammaire ambiguë)

Une grammaire $G$ est dite **ambiguë** s'il existe un mot de $L(G)$ ayant plusieurs arbres syntaxiques.

## Exemples : dérivations et arbres syntaxiques

### Example

Soit la grammaire $G = (\{S,\ T\},\ \{a,\ b,\ c\},\ \longrightarrow,\ S)$, définie par les règles
$S \longrightarrow aTb \mid c$
$T \longrightarrow cSS \mid S$

- $S \rightarrowtail aTb \rightarrowtail acSSb \rightarrowtail accSb \rightarrowtail accaSbb \rightarrowtail accacbb$
  est une dérivation à gauche (à chaque étape le non terminal le plus à gauche est réécrit)
- $S \rightarrowtail aTb \rightarrowtail acSSb \rightarrowtail acSaTbb \rightarrowtail accaTbb \rightarrowtail accaSbb \rightarrowtail accacbb$ est une dérivation quelconque

le mot *accacbb* a deux dérivations différentes et un arbre syntaxique.

## Représentation des arbres syntaxiques forme postfixée

**Definition**

Soit une grammaire $G = (N,\ T,\ \rightarrow,\ S)$ une grammaire algébrique, on ajoute un numéro de règle à chaque règle de la grammaire, on considére les représentation postfixée des arbres en faisant figurer les éléments de $T$ et les numéros des règles.

**Example**

Soit la grammaire $G = (\{S,\ T\},\ \{a,\ b,\ c\},\ \rightarrow,\ S)$, définie par les règles
$S \rightarrow aTb\ |1|\ c|2|$
$T \rightarrow cSS\ |3|\ S\ |4|$
l'arbre syntaxique du mot $accacbb$ s'écrit $acc2ac24b13b1$ sous forme postfixée

Avant de commencer l'analyse syntaxique d'une grammaire :
s'assurer que la grammaire est réduite (sinon la réduire)

## Analyse syntaxique descendante

- tentative de construire l'arbre syntaxique d'un mot du haut (c'est-à-dire de la racine (ou de l'axiome)) vers le bas (les feuilles (les nœuds étiquetés par les terminaux))
- tentative de trouver une dérivation à gauche pour le mot donné
- tentative de construire un arbre syntaxique pour le mot donné en construisant les nœuds de l'arbre en préordre
- détermination de classes de grammaires pour lesquelles il n'est pas utile de faire des retours arrières (les grammaires LL(1))

## Analyse descendante : exemples introductifs

On essaie de construire un arbre syntaxique du mot donné de haut en bas, c'est-à-dire en partant de l'axiome de la grammaire et en parcourant le mot donné de gauche à droite.

### Example (1)

$G = (\{S,\ T\},\ \{a,\ b,\ c,\ d\},\ \rightarrow,\ S)$ une grammaire définie par
$S \rightarrow aSbT \mid cT \mid d$
$T \rightarrow aT \mid bS \mid c$
on considère le mot $w = accbbadbc$

### Example (2)

$G = (\{S,\ A\},\ \{a,\ b,\ c,\ d\},\ \rightarrow,\ S)$ une grammaire définie par
$S \rightarrow aAb$
$A \rightarrow cd \mid c$
on considère le mot $w = acb$

### Example (3)

$G = (\{S\},\ \{a,\ b,\ c,\ d\},\ \rightarrow,\ S)$ une grammaire définie par
$S \rightarrow aSb \mid aSc \mid d$
on considère le mot $w = aaaaadbcbcb$

## Analyse descendante : exemples introductifs (suite)

### Example (4)

$G = (\{E,\ T,\ F,\ E',\ T'\},\ \{+,\ -,\ *,\ (,\ ),\ nb\ \},\ \},\ \rightarrow,\ E)$ une grammaire définie par
$E \rightarrow TE'$
$E' \rightarrow +TE' \mid -TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid /FT' \mid \varepsilon$
$F \rightarrow (E) \mid nb$
on considère le mot $w = 3 * 4 + 10 * (5 + 11)/34 + 12$

## Descente récursive

- écriture d'un programme qui étant une grammaire et un mot donné, détermine si le mot est engendré par la grammaire et si c'est le cas construit son arbre syntaxique
- à chaque non terminal de la grammaire correspond une procédure
- pour les terminaux on associe une procédure qui teste si le terminal lu est celui attendu
- cette démarche est utilisable pour les grammaires $LL(1)$ et adaptable aux grammaires $LL(k)$ $(k > 1)$

## Mise en œuvre

- Soit $G = (N,\ T,\ \rightarrow,\ S)$ une grammaire

  **Données** : un mot $\alpha$ de $T^*$ suivi du caractère \$, appelé marqueur de fin

  **Résultats** : un booléen *erreur* et une chaîne de caractères *res*,

  A la fin du programme si *erreur* = *vrai* alors $\alpha$ appartient à $L(G)$ et *res* est l'arbre syntaxique de $\alpha$
- Variables du programme : *res* : chaîne de caractères, *erreur* : booléen, *y* : le caractère courant du mot d'entrée $\alpha$
- Programme principal :

  *erreur* $\leftarrow$ *faux*;
  *res* $\leftarrow \varepsilon$;
  *lire*(*y*); –lecture du premier caractère de $\alpha$ qui est affecté à *y*

  *Ana-S* ; –appel de la procédure *Ana-S* correspondant à l'axiome de la grammaire

  si *erreur* alors "mot non engendré par la grammaire"
  sinon si *y* = \$ alors "mot engendré par la grammaire" , *res*
  sinon "mot non engendré par la grammaire"
  fsi
  fsi

- procédure associée à un terminal de $G$:

  Procédure Ana($x$ : $T$) ;
  si $\neg$*erreur* alors
  si $y = x$ alors
  debut
  $res \leftarrow res \oplus x$ ; *lire*($y$); $-\oplus$ concaténe deux chaînes de caractères
  fin
  sinon *erreur* $\leftarrow$ *vrai*
  fsi
  fsi

  Procédure qui teste si le caractère en argument est égal au caractère courant du mot donné.
- à chaque non terminal $X$, on associe une procédure *Ana-X* (voir exemple)

## Exemple

Soit la grammaire $G = (\{S,\ A\},\ \{a,\ b,\ c,\ d\},\ \rightarrow,\ S)$ où
$S \rightarrow aAb\ |1|$
$A \rightarrow cd\ |2|c\ |3|$

- procédure pour le non terminal $S$ : *Ana-S*

  procédure *Ana-S*;
  si $\neg$*erreur* alors
  debut
  *Ana*($a$); *Ana-A*; *Ana*($b$); *res* $\leftarrow$ *res* $\oplus$ 1
  fin
- procédure pour le non terminal $A$ : *Ana-A*

  procédure *Ana-A*;
  si $\neg$*erreur* alors
  debut
  *Ana*($c$);
  si $y = d$ alors
  debut *Ana*($d$) ; *res* $\leftarrow$ *res* $\oplus$ 2 fin
  sinon si $y = b$ alors
  *res* $\leftarrow$ *res* $\oplus$ 3
  sinon *erreur* $\leftarrow$ *vrai*
  fsi
  fsi
  fin

## "l'approche descente récursive marche-t-elle dans tous les cas?"

**Problèmes :**

- Règles de la forme $X \longrightarrow X_1\alpha_1$ et $X \longrightarrow X_2\alpha_2$
  quelle est la règle à appliquer : idée : connaître par quelle lettre peut commencer $X_1$ et $X_2$
  Idée : notion de *Premier*
  et si $X_1 \overset{*}{\longmapsto} \varepsilon$?
- Règles de la forme $X \longrightarrow \alpha$ et $X \longrightarrow \varepsilon$
  quelle règle à appliquer : idée : connaître par quelle lettre peut commencer $\alpha$ (notion de *Premier*) et par quelle lettre peut être suivi $X$ dans une dérivation (notion de *Suivant*)

## Non terminaux produisant le vide

### Definition

Soit $G = (N,\ T,\ \rightarrow,\ S)$ une grammaire algébrique et $X \in N$, $Vide = \{X\ ;\ X \in N \text{ et } X \overset{*}{\rightarrowtail} \varepsilon\}$.
*Vide* est l'ensemble des terminaux qui produisent $\varepsilon$

### Example

$G = (\{E,\ T,\ F,\ E',\ T'\},\ \{+,\ -,\ *,\ (,\ ),\ nb\ \},\ \},\ \rightarrow,\ E)$ une grammaire définie par
$E \rightarrow TE'$
$E' \rightarrow +TE' \mid -TE' \mid \varepsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid /FT' \mid \varepsilon$
$F \rightarrow (E) \mid nb$

$Vide = \{E',\ T'\}$

## Définition des premiers

### Definition (Premiers)

Soit $G = (N,\ T,\ \rightarrow,\ S)$ une grammaire algébrique et $\alpha \in (N \cup T)^*$,
$Premier(\alpha) = \{x\ ;\ x \in T \text{ et } \alpha \overset{*}{\rightarrowtail} xw\}$.

### Calcul des premiers pour les non terminaux

1. pour tout non terminal $X$ de la grammaire $G$, initialiser $Premier(X)$ à l'ensemble vide fpour
2. pour toute règle de la forme $X \rightarrow Y_1 \ldots Y_n$
   si $Y_1 \in T$ alors -ajouter $Y_1$ à $Premier(X)$
   sinon -ajouter $Premier(Y_1)$ à $Premier(X)$;
   -pour tout $j \in \{2, \ldots, n\}$ tq $\forall i \in \{1, \ldots, j\text{-}1\}$ tq $Y_i \in Vide$
   ajouter $Premier(Y_j)$ à $Premier(X)$
   fpour
   fsi

   fpour
3. recommencer l'étape 2 jusqu'à ce qu'il n'y ait plus de changement kgh

## Exemple(1) : calcul des premiers

**Example (Calcul des premiers (1))**

$G = (\{E,\ T,\ F,\ E',\ T'\},\ \{+,\ -,\ *,\ (,\ ),\ nb\ \},\ \},\ \rightarrow,\ E)$ la grammaire définie par

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

on a $Vide = \{E',\ T'\ \}$

| $Premier(E)$ | $Premier(E')$ | $Premier(T)$ | $Premier(T')$ | $Premier(F)$ |
|---|---|---|---|---|
| $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $\emptyset$ | $\{+,\ -\}$ | $\emptyset$ | $\{*,\ /\}$ | $\{(,\ nb\}$ |
| $\emptyset$ | $\{+,\ -\}$ | $\{(,\ nb\}$ | $\{*,\ /\}$ | $\{(,\ nb\}$ |
| $\{(,\ nb\}$ | $\{+,\ -\}$ | $\{(,\ nb\}$ | $\{*,\ /\}$ | $\{(,\ nb\}$ |
| *idem* | *idem* | *idem* | *idem* | *idem* |
| | | | | |

## Exemple(2) : calcul des premiers

**Example (Calcul des premiers (1))**

$G = (\{S,\ A,\ B,\ C\},\ \{a,\ b,\ c,\ d,\ e\},\ \rightarrow,\ S\})$ une grammaire définie par

$S \rightarrow ABCe$
$A \rightarrow aA \mid \varepsilon$
$B \rightarrow bB \mid cB \mid \varepsilon$
$C \rightarrow de \mid da \mid dA$

on a $Vide = \{A,\ B\}$

| $Premier(S)$ | $Premier(A)$ | $Premier(B)$ | $Premier(C)$ |
|---|---|---|---|
| $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| $\emptyset$ | $\{a\}$ | $\{b,\ c\}$ | $\{d\}$ |
| $\{a,\ b,\ c,\ d\}$ | $\{a\}$ | $\{b,\ c\}$ | $\{d\}$ |
| *idem* | *idem* | *idem* | *idem* |

## Extension : définition des premiers pour les éléments de $(N \cup T)^*$

### Definition

Soit $\alpha \in (N \cup T)^*$, on étend la fonction $Premier(\alpha)$ aux éléments de $(N \cup T)^*$ de la façon suivante :

- $Premier(a\beta) = \{a\}$ si $a \in T$
- $Premier(X)$ est déjà défini pour $X \in N$
- $Premier(X\beta) =$ si $X \notin Vide$ alors $Premier(X)$ sinon $Premier(X) \cup Premier(\beta)$ fsi où $\beta \neq \varepsilon$

## Définition des suivants

### Definition (Suivants)

Soit la grammaire $G = (N,\ T,\ \longrightarrow,\ S)$ et soit $A \in N$, $Suivant(A)$ est l'ensemble des éléments $a$ de $T \cup \{\$\}$ qui peuvent apparaître immédiatement après $A$ dans une dérivation (c'est-à-dire les éléments $a$ tel que $S \overset{*}{\longmapsto} \alpha A a \beta$)

### Calcul des suivants des symboles non-terminaux

1. initialiser $Suivant(S)$ à $\{\$\}$;
   pour tout non terminal $X \neq S$ de la grammaire $G$, initialiser $Suivant(X)$ à l'ensemble vide
   fpour
2. pour chaque règle de la forme $A \longrightarrow \alpha B \beta$ où $B \in N$ ajouter $Premier(\beta)$ à $Suivant(B)$
3. pour chaque règle $A \longrightarrow \alpha B$, ajouter $Suivant(A)$ à $Suivant(B)$
4. pour chaque règle $A \longrightarrow \alpha B \beta$ tel que $\beta \overset{*}{\longmapsto} \varepsilon$ ajouter $Suivant(A)$ à $Suivant(B)$

recommencer à partir de l'étape 3 jusqu'à ce que l'on n'ajoute rien de nouveau dans les ensembles *Suivant*

## Exemple de calcul de suivants

### Example (1)

$G = (\{E,\ T,\ F,\ E',\ T'\},\ \{+,\ -,\ *,\ (,\ ),\ nb\ \},\ \},\ \rightarrow,\ E)$ la grammaire définie par

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

$Vide = \{E',\ T'\ \}$

$Premier(E) = \{(,\ nb\}$, $Premier(E') = \{+,\ -\}$, $Premier(T) = \{(,\ nb\}$, $Premier(T') = \{*,\ /\}$ et $Premier(F) = \{(,\ nb\}$

| $Suivant(E)$ | $Suivant(E')$ | $Suivant(T)$ | $Suivant(T')$ | $Suivant(F)$ |
|---|---|---|---|---|
| \$ | $\emptyset$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |
| \$ ) | \$ | + − \$ | + − \$ | * / + − \$ |
| \$ ) | \$ ) | + − \$ ) | + − \$ ) | * / + − \$ ) |

## Conditions de terminaison des procédures de la descente récursive

**Théorème**

Les procédures d'analyse syntaxique d'une grammaire $G$ s'arrêtent pour toute donnée $\alpha$ si et seulement si la grammaire n'est pas récursive gauche.

C'est-à-dire s'il n'existe pas de non terminal $A$ vérifiant $A \overset{+}{\rightarrowtail} A\beta$.

Idée de démonstration : les procédures d'analyse syntaxique ne terminent pas, si le texte en entrée n'est plus consommé à partir d'un certain temps. Cela ne peut se produire que dans le cas où l'on a des règles de la forme suivante :
$A_1 \rightarrow A_2\alpha_2$, $A_2 \rightarrow A_3\alpha_3$, ..., $A_k \rightarrow A_{k+1}\alpha_{k+1}$, ...
où les $A_i$ sont des non terminaux.

Comme les $A_i$ sont finis, il existe un $k$ suffisamment grand tel que $A \overset{*}{\rightarrowtail} A\beta$

**Remarque :** cette condition est une condition générale nécessaire à toutes les démarches concernant l'analyse descendante.

## Construction de la table d'analyse LL

Soit $G = (N, T, \rightarrow, S)$ une grammaire, une table d'analyse $M$ pour la grammaire $G$ est une table à deux entrées.
Pour chaque élément $A$ de $N$ et chaque élément $a$ de $T \cup \{\$\}$, $M[A,\ a]$ indique la règle de la grammaire à appliquer.

### Construction de la table d'analyse

- Pour chaque règle $A \rightarrow \alpha$ de la grammaire faire
  1. pour tout $a \in Premier(\alpha)$, ajouter la règle $A \rightarrow \alpha$ dans la case $M[A,\ a]$
  2. si $\alpha \overset{*}{\rightarrowtail} \varepsilon$ alors pour tout $b \in Suivant(A)$ ajouter la règle $A \rightarrow \alpha$ dans la case $M[A,\ b]$
- chaque case vide de $M$ correspond à une erreur de syntaxe

## Exemple de construction d'une table d'analyse

### Example

$G = (\{E,\ T,\ F,\ E',\ T'\},\ \{+,\ -,\ *,\ (,\ ),\ nb\ \},\ \},\ \rightarrow,\ E)$ la grammaire définie par

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

- $Vide = \{E',\ T'\ \}$
- $Premier(E) = \{(,\ nb\}$, $Premier(E') = \{+\}$, $Premier(T) = \{(,\ nb\}$, $Premier(T') = \{*\}$ et $Premier(F) = \{(,\ nb\}$
- $Suivant(E) = \{\$,\ )\}$, $Suivant(E') = \{\$,\ )\}$, $Suivant(T) = \{+,\ \$,\ )\}$, $Suivant(T') = \{+,\ \$,\ )\}$, $Suivant(F) = \{*,\ +,\ \$,\ )\}$

**Table d'analyse LL**

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

## Grammaire LL(1)

**Definition**

On appelle grammaire LL(1) une grammaire pour laquelle toutes les cases de la table d'analyse contiennent au plus une règle de la grammaire.

Dans l'abréviation LL(1) :

- le premier L signifie que le mot donné en entrée est lu de gauche à droite (Left to right scanning)
- le second L signifie que la méthode recherche une dérivation à gauche, c'est-à-dire que l'on cherche à réécrire le non terminal le plus à gauche (Leftmost derivation)
- le 1 signifie qu'en lisant 1 caractère à l'avance on est capable de dire quelle règle utiliser

**Example**

La grammaire du transparent précédent définissant les expressions arithmétiques est une grammaire LL(1).

## Symboles directeurs d'une règle

On utilise parfois la notion de symbole directeur d'une règle.

**Definition (Symbole directeur d'une règle)**

L'ensemble des symboles directeurs de $A \longrightarrow \alpha$ est défini par

$SymbolesDirecteurs(A \longrightarrow \alpha) =$
si $non(\alpha \overset{*}{\rightarrowtail} \varepsilon)$
alors $Premier(\alpha)$
sinon $Premier(\alpha) \cup Suivant(A)$

- lors de la construction de la table d'analyse $M$ : si $d \in SymbolesDirecteurs(A \longrightarrow \alpha)$ alors on ajoute $A \longrightarrow \alpha$ dans $M[A,\ d]$.
- pour que la grammaire soit LL(1) il faut et il suffit que pour tout couple de règles différentes de la forme $A \longrightarrow \alpha$ et $A \longrightarrow \beta$ on ait :

$$SymbolesDirecteurs(A \longrightarrow \alpha) \cap SymbolesDirecteurs(A \longrightarrow \beta) = \emptyset$$

## Analyseur syntaxique

On suppose que la grammaire est LL(1) et que M est la table d'analyse de la grammaire.
On utilise une pile, à l'initialisation la pile contient $\$S$ (le sommet de la pile est $S$ l'axiome de la grammaire).
Variables du programme :

- X : le symbole en sommet de pile
- a : le symbole (terminal) courant du mot analysé
- erreur : booléen (initialisé à faux)
- accepter : booléen (initialisé à faux)

## Analyseur syntaxique : algorithme

répeter
  $X \leftarrow$ sommet de pile
  $a \leftarrow$ caractére du mot à analyser
  si $X$ est un non terminal alors
    si $M[X, a] = X \rightarrow Y_1 \ldots Y_n$
      émettre en sortie la règle $X \rightarrow Y_1 \ldots Y_n$
      dépiler $X$ de la pile;
      empiler $Y_n, \ldots, Y_1$;
    sinon *erreur* $\leftarrow$ *vrai* – la case est vide dans la table
    fsi
  sinon – $X$ est un terminal
     si $X =$ \$ alors – la pile est vide
      si $a =$ \$ alors *accepter* $\leftarrow$ *vrai* – pile vide et caractère de fin de mot
      sinon *erreur* $\leftarrow$ *vrai*
      fsi
    sinon – la pile n'est pas vide
      si $X = a$ alors – l'élément en sommet de pile est le caractère courant du mot
        depiler $X$;
        lire le caractère suivant du mot donné;
      sinon – l'élément en sommet de pile est différent du caractère courant du mot
        *erreur* $\leftarrow$ *vrai*
      fsi
    fsi
  fsi

jusqu'à erreur ou accepter

## Exemple d'exécution de l'analyseur syntaxique

Grammaire des expressions arithmétiques, mot : $2 + 5 * 7$

| *PILE* | *Entrée* | *Sortie* |
|---|---|---|
| $\$E$ | $2 + 5 * 7\$$ | $E \rightarrow TE'$ |
| $\$E'T$ | $2 + 5 * 7\$$ | $T \rightarrow FT'$ |
| $\$E'T'F$ | $2 + 5 * 7\$$ | $F \rightarrow 2$ |
| $\$E'T'2$ | $2 + 5 * 7\$$ | |
| $\$E'T'$ | $+5 * 7\$$ | $T' \rightarrow \varepsilon$ |
| $\$E'$ | $+5 * 7\$$ | $E' \rightarrow +TE'$ |
| $\$E'T+$ | $+5 * 7\$$ | |
| $\$E'T$ | $5 * 7\$$ | $T \rightarrow FT'$ |
| $\$E'T'F$ | $5 * 7\$$ | $F \rightarrow 5$ |
| $\$E'T'5$ | $5 * 7\$$ | |
| $\$E'T'$ | $*7\$$ | $T' \rightarrow *FT'$ |
| $\$E'T'F*$ | $*7\$$ | |
| $\$E'T'F$ | $7\$$ | $F \rightarrow 7$ |
| $\$E'T'7$ | $7\$$ | |
| $\$E'T'$ | $\$$ | $T' \rightarrow \varepsilon$ |
| $\$E'$ | $\$$ | $E' \rightarrow \varepsilon$ |
| $\$$ | $\$$ | *accepter* (*le mot est engendré par la grammaire*) |

## Propriétés des grammaires LL(1)

### Proposition

Une grammaire $G$, LL(1), possède les propriétés suivantes :

- s'il existe deux règles $A \longrightarrow \alpha$ et $A \longrightarrow \beta$ de $G$ :
  1. $Premier(\alpha) \cap Premier(\beta) = \emptyset$
  2. au plus un de $\alpha$ ou $\beta$ vérifie $\alpha \overset{*}{\rightarrowtail} \varepsilon$ ou $\beta \overset{*}{\rightarrowtail} \varepsilon$
  3. si $\beta \overset{*}{\rightarrowtail} \varepsilon$ alors $Premier(\alpha) \cap Suivant(A) = \emptyset$
- la grammaire $G$ n'est pas ambiguë.
- la grammaire $G$ ne comporte pas de récursivité à gauche (l'algorithme ne termine pas pour des grammaires comportant une récursivité gauche).

## Récusrivité à gauche

**Definition (Récursivité gauche immédiate)**

Une grammaire est **immédiatement récursive à gauche** s'il existe un non terminal $A$ et une règle de la forme $A \longrightarrow A\alpha$ où $\alpha$ est une chaîne de terminaux ou non terminaux quelconques.

**Example**

La grammaire suivante comporte plusieurs récursivité gauches immédiates :

$S \longrightarrow ScA \mid B$
$A \longrightarrow Aa \mid \varepsilon$
$B \longrightarrow Bb \mid d \mid e$

## Suppression de la récursivité à gauche immédiate

### Proposition

On remplace des règles de la forme

$$A \longrightarrow A\alpha_1 \mid \ldots \mid A\alpha_k \mid \beta_1 \mid \ldots \mid \beta_p$$

par les règles suivantes :

$$\begin{array}{l} A \longrightarrow \beta_1 A' \mid \ldots \mid \beta_p A' \\ A' \longrightarrow \alpha_1 A' \mid \ldots \mid \alpha_k A' \mid \varepsilon \end{array}$$

**Démonstration** : cela provient du fait que l'on a $A \overset{*}{\rightarrowtail} \{\beta_1, \ldots, \beta_k\}\{\alpha_1, \ldots, \alpha_k\}^*$

## Exemple de suppression de la récursivité à gauche immédiate

### Example

$S \longrightarrow ScA \mid B$
$A \longrightarrow Aa \mid \varepsilon$
$B \longrightarrow Bb \mid d \mid e$

$S \longrightarrow BS'$
$S' \longrightarrow cAS' \mid \varepsilon$
$A \longrightarrow A'$
$A' \longrightarrow aA' \mid \varepsilon$
$B \longrightarrow dB' \mid eB'$
$B' \longrightarrow bB' \mid \varepsilon$

## Récursivité à gauche

### Definition (Récursivité à gauche)

Une grammaire est récursive à gauche s'il existe un non terminal $A$ et une dérivation de la forme $A \overset{+}{\rightarrowtail} A\alpha$.

### Example

$S \longrightarrow Aa \mid b \qquad A \longrightarrow Ac \mid Sd \mid c$

Le non terminal $S$ est récursif à gauche car $S \rightarrowtail Aa \rightarrowtail Sda$ (mais $S$ n'est pas immédiatement récursif à gauche).

### Suppression de la récursivité à gauche pour des grammaires sans règle $A \longrightarrow \varepsilon$ et sans cycle

**Hypothèses :** on suppose que la grammaire donnée ne possède pas de règles $A \longrightarrow \varepsilon$ ni de cycle (c'est-à-dire qu'il n'existe pas $A$ tel que $A \overset{+}{\rightarrowtail} A$)

Ordonner les non terminaux de la grammaire $A_1, \ldots, A_n$
pour $i = 1$ à $n$ faire
  pour $j = 1$ à $i$-1 faire
    remplacer chaque règle de la forme $A_i \longrightarrow A_j\alpha$ où $A_j \longrightarrow \beta_1 \mid \ldots \mid \beta_p$
    par $A_i \longrightarrow \beta_1\alpha \mid \ldots \mid \beta_p\alpha$
  fpour
  éliminer les récursivités à gauche immédiates des règles dont les membres gauches sont $A_i$
fpour

## Exemple de suppression de récursivité

### Example

$S \longrightarrow Aa \mid b$
$A \longrightarrow Ac \mid Sd \mid c$
On ordonne les non terminaux : $A_1 = S$, $A_2 = A$

1. $i = 1$ (la boucle interne est vide et il n'y a pas de récursivité immédiate de $S$)
   $S \longrightarrow Aa \mid b$
2. $i = 2$
   on remplace $A \longrightarrow Sd$ par $A \longrightarrow Aad \mid bd$,
   on obtient ainsi $A \longrightarrow Ac \mid Aad \mid bd \mid c$
   On élimine la récursivité immédiate à gauche de $A$
   $A \longrightarrow bdA' \mid cA'$
   $A' \longrightarrow cA' \mid adA' \mid \varepsilon$

Le résultat est la grammaire suivante :
$S \longrightarrow Aa \mid b$
$A \longrightarrow bdA' \mid cA'$
$A' \longrightarrow cA' \mid adA' \mid \varepsilon$

## Contre-exemple de suppression de la récursivité à gauche

### Example

$S \longrightarrow Sa \mid TSc \mid d$
$T \longrightarrow SbT \mid \varepsilon$
On ordonne les non terminaux : $A_1 = S$, $A_2 = T$,

1. $i = 1$
   On supprime la récursivité immédiate à gauche de $S$
   $S \longrightarrow TScS' \mid dS'$
   $S' \longrightarrow aS' \mid \varepsilon$
2. $i = 2$
   On remplace $T \longrightarrow SbT$ par
   $T \longrightarrow TScS'bT \mid dS'bT$
   On obtient donc $T \longrightarrow TScS'bT \mid dS'bT \mid \varepsilon$
   On élimine la récursivité immédiate à gauche de $T$
   $T \longrightarrow dS'bTT' \mid T'$
   $T' \longrightarrow ScS'bTT' \mid \varepsilon$

On obtient la grammaire suivante :
$S \longrightarrow TScS' \mid dS'$
$S' \longrightarrow aS' \mid \varepsilon$
$T \longrightarrow dS'bTT' \mid T'$
$T' \longrightarrow ScS'bTT' \mid \varepsilon$
On n'a pas supprimé la récursivité gauche car
$S \rightarrowtail TScS' \rightarrowtail T'ScS' \rightarrowtail ScS'$

## Factorisation à gauche

Lorsque deux règles sont de la forme $A \longrightarrow \alpha\beta_1$ ou $A \longrightarrow \alpha\beta_2$, il n'est en général pas possible de déterminer quelle règle on va utiliser en lisant un caractère à l'avance. L'idée est donc de différer la décision jusqu'à ce qu'on ait lu suffisamment de texte.

### Example

$S \longrightarrow abcdAab \mid abcdBbd$
$A \longrightarrow aC$
$B \longrightarrow bD$
$C \longrightarrow \ldots$
$D \longrightarrow \ldots$

Il faut lire le 5ième caractère à l'avance pour déterminer quelle règle choisir entre $S \longrightarrow abcdAab$ et $S \longrightarrow abcdBbd$, la grammaire n'est pas LL(1).

## Factorisation : algorithme

### Algorithme

pour chaque non terminal A
trouver le plus long préfixe commun $\alpha$ à deux ou plus de deux membres droits de règles
si $\alpha \neq \varepsilon$, remplacer $A \rightarrow \alpha\beta_1 \mid \ldots \mid \alpha\beta_n \mid \gamma_1 \mid \ldots \mid \gamma_p$ (où $\alpha$ n'est pas préfixe de $\gamma_i$) par
$A \rightarrow \alpha A' \mid \gamma_1 \mid \ldots \mid \gamma_p$
$A' \rightarrow \beta_1 \mid \ldots \mid \beta_n$
Recommencer jusqu'à ce qu'il n'y ait plus de factorisation à effectuer

### Example

$S \rightarrow aAbS \mid aAbSeB \mid a$
$A \rightarrow bcB \mid bca$
$B \rightarrow ab$

**Exécution de l'algorithme**

- pour $S$
  - le plus long préfixe est $aAbS$, on obtient
    $S \rightarrow aAbSS' \mid a$
    $S' \rightarrow eB \mid \varepsilon$
  - le plus long préfixe est $a$, on obtient
    $S \rightarrow aS''$
    $S'' \rightarrow AbSS' \mid \varepsilon$
- pour $A$ le plus long préfixe est $bc$, on obtient

## Factorisation

La grammaire obtenue par factorisation est :

$S \longrightarrow aS''$
$S'' \longrightarrow AbSS' \mid \varepsilon$
$S' \longrightarrow eB \mid \varepsilon$
$A \longrightarrow bcA'$
$A' \longrightarrow B \mid a$
$B \longrightarrow ab$

## Conclusion

Etant donnée une grammaire

- Réduire la grammaire (en éliminant les règles et les symboles inutiles)
- Rendre la grammaire non ambiguë si nécessaire (il n'y a pas d'algorithme pour déterminer si une grammaire est ambiguë et pour en trouver une non ambiguë)
- Eliminer la récursivité à gauche si nécessaire
- Factoriser la grammaire si nécessaire
- Construire la table d'analyse (après avoir calculé *Premier* et *Suivant*)

Si la grammaire n'est pas LL(1), utiliser une autre méthode pour l'analyse syntaxique.

Il existe des méthodes qui généralisent LL(1). On peut caractériser les grammaires LL(k) en généralisant les définitions de *Premier* et *Suivant*.

Les méthodes d'analyse ascendante seront vues en deuxième année, dans le module de Traduction.