

Grammaires et Analyse Syntaxique - Cours 5 Analyse LL(1) dans le cas général

Ralf Treinen



treinen@irif.fr

20 février 2025

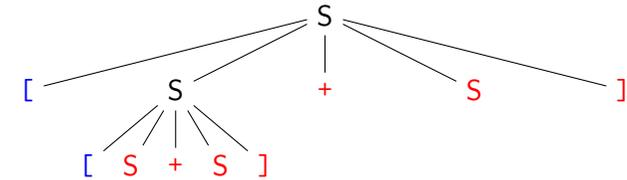
© Ralf Treinen 2020–2025

Grammaires LL(1)

- ▶ Intuition derrière les grammaires LL(1) : dans la construction d'une dérivation gauche, le symbole suivant de l'entrée nous indique quelle règle de la grammaire appliquer au non-terminal le plus à gauche de l'arbre de dérivation qui reste à développer.
- ▶ Conséquence : toute grammaire LL(1) (ou même LL(k)) est non-ambiguë.

Construction d'un arbre de dérivation

Règles de la grammaire : (1) $S \rightarrow i$ (2) $S \rightarrow [S + S]$



Choisir règle (1) : c'est la seule qui peut produire à partir de S un mot qui commence par i .

Rappel First_1

- ▶ Étant donnée une grammaire $G = (\Sigma, N, S, P)$ et un mot $\alpha \in (N \cup \Sigma)^*$

$$\text{First}_1(\alpha) = \{c \in \Sigma \mid \exists w \in \Sigma^*, \alpha \rightarrow^* cw\}$$

- ▶ $\text{First}_1(\alpha)$ est l'ensemble des symboles par lesquels un mot terminal dérivé à partir de α peut commencer.
- ▶ *En absence de règles* $N \rightarrow \epsilon$: une grammaire est LL(1) ssi pour toutes les règles différentes $N \rightarrow \alpha$ et $N \rightarrow \beta$ pour le *même* non-terminal

$$\text{First}_1(\alpha) \cap \text{First}_1(\beta) = \emptyset$$

Calcul de First_1 dans le cas simple

- ▶ Le cas simple est : tout côté droit des règles de la grammaire commence par un *terminal*.
- ▶ Dans ce cas on a simplement : $\text{First}_1(c\alpha) = \{c\}$ où $c \in \Sigma$.
- ▶ Exemple :

$$S \rightarrow i \quad S \rightarrow [S + S]$$

Cette grammaire est LL(1).

L'exemple des expressions partiellement parenthésées

- ▶ Cette grammaire est non-ambiguë ☺
- ▶ Intuition : Les "+" peuvent être produits seulement à partir du E. Toute somme engendrée par E "au-dessous" d'un * est protégée par des parenthèses (et).
- ▶ Argument plus précis : voir dans 2 semaines.
- ▶ Arbres de dérivation pour

$$i+i*i \text{ EOF} \quad \text{et} \quad i+i+i \text{ EOF}$$

- ▶ Cette grammaire est-elle aussi LL(1) ?
- ▶ Réponse non, on verra dans la suite pourquoi pas.

Deuxième grammaire pour les expressions parenthésées

- ▶ La grammaire $G_2 = (\{i, +, *, (,), \text{EOF}\}, \{E, T, F\}, S, P)$, où P est

$$S \rightarrow E \text{ EOF}$$

$$E \rightarrow E+T \mid T$$

$$T \rightarrow T*F \mid F$$

$$F \rightarrow (E) \mid i$$

- ▶ Cette grammaire décrit les expressions arithmétiques partiellement parenthésées.

Calcul de First_1 pour les non-terminaux

- ▶ Grammaire $G = (\Sigma, N, S, P)$.
Hypothèse : aucune production $N \rightarrow \epsilon$.
- ▶ On construit un graphe, avec N comme ensemble de nœuds.
- ▶ On met une arête de A vers B quand il y a dans P une production de la forme $B \rightarrow A\alpha$.
- ▶ Il est inutile de mettre une arête d'un nœud vers lui-même.
- ▶ On ajoute des symboles de Σ comme valeurs aux nœuds. Initialement, on ajoute à un nœud A la valeur a quand il y a dans P une production $A \rightarrow a\alpha$.
- ▶ Puis on propage les valeurs dans le sens des flèches, jusqu'à ce qu'on ne puisse plus rien propager.

Exemple

- Grammaire $G = (\{a, (,), +\}, \{F, S\}, S, P)$ où P est

$$F \rightarrow a$$

$$S \rightarrow (F+S)$$

$$S \rightarrow F$$

► Initialisation :
$$F \xrightarrow{\quad} S$$

$$\{a\} \quad \{(\}$$

► Propagation :
$$F \xrightarrow{\quad} S$$

► Résultat : $\text{First}_1(F) = \{a\}$, $\text{First}_1(S) = \{(, a\}$.

Exemple (2)

- Grammaire $G = (\{F, S\}, \{a, (,), +\}, S, P)$ où P est

$$F \rightarrow a$$

$$S \rightarrow (F+S)$$

$$S \rightarrow F$$

- Nous avons déjà : $\text{First}_1(F) = \{a\}$, $\text{First}_1(S) = \{(, a\}$.

- On obtient donc :

$$\text{First}_1(a) = \{a\}$$

$$\text{First}_1((F+S)) = \{(}$$

$$\text{First}_1(F) = \{a\}$$

Calcul de First_1 sur des séquences de symboles

- Toujours sous l'hypothèse qu'on n'a pas de règle $N \rightarrow \epsilon$
- On étend la fonction First_1 à des séquences *non vides* de symboles :

$$\text{First}_1(x\alpha) = \begin{cases} \{x\} & \text{si } x \in \Sigma \\ \text{First}_1(x) & \text{si } x \in N \end{cases}$$

- Nous allons utiliser cette généralisation pour calculer les First_1 des côtés droits des règles de la grammaire.

Retour à l'exemple de la grammaire G_2

- Règles :

$$S \rightarrow E \text{ EOF}$$

$$E \rightarrow E+T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid i$$

- Construction au tableau.

Que faire ?

- ▶ On peut transformer la grammaire en une grammaire équivalente (qui définit le même langage), et qui est LL(1).
- ▶ Ce n'est pas toujours possible, et il y a deux inconvénients :
 - ▶ la grammaire résultante peut être plus grande ;
 - ▶ la structure de l'arbre de dérivation peut changer.
- ▶ Il y a un troisième inconvénient : la transformation peut introduire des règles $K \rightarrow \epsilon$, il faut donc adapter la technique à ce cas.

Explication de la transformation

- ▶ Les deux règles originales pour le non-terminal E :

$$\begin{aligned} E &\rightarrow T \\ E &\rightarrow E+T \end{aligned}$$

- ▶ Dans la grammaire d'origine, le non-terminal E engendre une séquence non-vide de non-terminaux T, séparés par des +.
- ▶ Dans la grammaire transformée, le non-terminal E' engendre la suite de cette séquence après un T :

$$\begin{aligned} E &\rightarrow T E' \\ E' &\rightarrow \epsilon \mid +E \end{aligned}$$

- ▶ Pareil pour le non-terminal T.

La grammaire transformée

- ▶ Une grammaire pour les expressions arithmétiques *partiellement* parenthésées.
- ▶ Terminaux : $\{i, +, *, (,), \text{EOF}\}$

grammaire originale	grammaire transformée
$S \rightarrow E \text{ EOF}$	$S \rightarrow E \text{ EOF}$
$E \rightarrow E+T \mid T$	$E \rightarrow T E'$
	$E' \rightarrow \epsilon \mid +E$
$T \rightarrow T * F \mid F$	$T \rightarrow F T'$
	$T' \rightarrow \epsilon \mid *T$
$F \rightarrow (E) \mid i$	$F \rightarrow (E) \mid i$

- ▶ Axiome : S

Non-terminaux annulables

Définition

Soit $G = (\Sigma, N, S, P)$ une grammaire. $K \in N$ est *annulable* si $K \rightarrow^* \epsilon$.

On note souvent EPS l'ensemble des non-terminaux annulables d'une grammaire.

Calcul des non-terminaux annulables

- ▶ Si $K \rightarrow \epsilon \in P$ alors K est annulable.
- ▶ Si $K \rightarrow K_1 \dots K_n \in P$, et K_i est annulable pour tout i , alors K est aussi annulable.

Exemple

- ▶ Grammaire $(\{a, b, c\}, \{A, B, C, D, E, F\}, F, P)$ avec P :

$$\begin{array}{l} A \rightarrow \epsilon \mid a \quad C \rightarrow A B \mid c \quad E \rightarrow A B C \\ B \rightarrow \epsilon \mid b \quad D \rightarrow C a C \quad F \rightarrow E d \mid C F \end{array}$$

- ▶ Sont annulables :
 - ▶ A, B car il y a ϵ côté droit
 - ▶ C car A, B annulables
 - ▶ E car A, B, C annulables
- ▶ D et F ne sont pas annulables.

Comment calculer First_1 dans le cas général ?

- ▶ Imaginez une règle $A \rightarrow B C d E$
- ▶ Soit EPS l'ensemble des non-terminaux annulant.
- ▶ Si $B \notin EPS$: dans cette règle, seulement $\text{First}_1(B)$ peut contribuer à $\text{First}_1(A)$.
- ▶ Si $B \in EPS$: $\text{First}_1(C)$ peut aussi contribuer à $\text{First}_1(A)$.
- ▶ Si $B \in EPS$ et $C \in EPS$: d doit être dans $\text{First}_1(A)$.
- ▶ Dans aucun des cas, $\text{First}_1(E)$ ne peut contribuer car il se trouve derrière le terminal d .

Calcul de First_1 avec non-terminaux annulables

- ▶ On construit un graphe, où les nœuds sont les non-terminaux.
- ▶ On met une arête de A vers B quand il y a une règle $B \rightarrow K_1 \cdots K_n A \alpha$ où $n \geq 0$, et tous les K_i sont annulables.
- ▶ Initialement on met sur un nœud K tous les terminaux a tel qu'il existe une règle $K \rightarrow K_1 \cdots K_n a \alpha$ où tous les K_i sont annulables.
- ▶ Puis on propage les valeurs dans le sens des flèches.

Calcul de First_1 sur l'exemple

$$\begin{array}{l} E \rightarrow T E' \quad E' \rightarrow \epsilon \mid +E \\ T \rightarrow F T' \quad T' \rightarrow \epsilon \mid *T \\ F \rightarrow (E) \mid i \quad S \rightarrow E \text{ EOF} \end{array}$$

- ▶ Non-terminaux annulables : E', T'

- ▶ Initialisation :

$$\begin{array}{cccccc} \{i, (\} & \{\} & \{\} & \{\} & \{*\} & \{+\} \\ F \longrightarrow & T \longrightarrow & E \longrightarrow & S & T' & E' \end{array}$$

- ▶ Propagation :

$$\begin{array}{cccccc} \{i, (\} & \{i, (\} & \{i, (\} & \{i, (\} & \{*\} & \{+\} \\ F \longrightarrow & T \longrightarrow & E \longrightarrow & S & T' & E' \end{array}$$

Calcul de $\text{First}_{\leq 1}$ dans le cas général

- ▶ On calcule maintenant pour les côtés droits de la grammaire
 $\text{First}_{\leq 1}(\alpha) := \{w : 1 \mid \alpha \rightarrow^* w, w \in \Sigma^*\}$
- ▶ La différence avec $\text{First}_1(\alpha)$ est que $\text{First}_{\leq 1}(\alpha)$ peut aussi contenir ϵ .
- ▶ $\text{First}_{\leq 1}(\epsilon) = \{\epsilon\}$
- ▶ pour tout $a \in \Sigma$: $\text{First}_{\leq 1}(a\alpha) = \{a\}$
- ▶ pour tout $A \in N$:

$$\text{First}_{\leq 1}(A\alpha) = \begin{cases} \text{First}_1(A) & \text{si } A \notin \text{EPS} \\ \text{First}_1(A) \cup \text{First}_{\leq 1}(\alpha) & \text{si } A \in \text{EPS} \end{cases}$$

Calcul de $\text{First}_{\leq 1}$ des côtés droits dans l'exemple

$$\begin{array}{l} E \rightarrow T E' \quad E' \rightarrow \epsilon \mid +E \\ T \rightarrow F T' \quad T' \rightarrow \epsilon \mid *T \\ F \rightarrow (E) \mid i \quad S \rightarrow E \text{ EOF} \end{array}$$

A	$\text{First}_1(A)$
S	{i, (}
E	{i, (}
E'	{+}
T	{i, (}
T'	{*}
F	{i, (}

$$\text{EPS} = \{E', T'\}$$

α	$\text{First}_{\leq 1}(\alpha)$
T E'	{i, (}
ϵ	{ ϵ }
+ E	{+}
F T'	{i, (}
* T	{*}
(E)	{(}
i	{i}
E EOF	{i, (}

Nous avons besoin de plus d'information !

- ▶ Donc on sait maintenant calculer $\text{First}_{\leq 1}$ aussi en présence de symboles annulables.
- ▶ Mais : En présence de symboles annulables, le fait que les $\text{First}_{\leq 1}$ des côtés droits pour le même non-terminal sont disjoints, n'est pas suffisant pour conclure que la grammaire est LL(1).
- ▶ Quand on a, dans l'arbre de dérivation, un non-terminal N à développer, et quand on a des symboles annulables, alors le terminal suivant de l'entrée ne vient pas nécessairement du N !

Nous avons besoin de plus d'information !

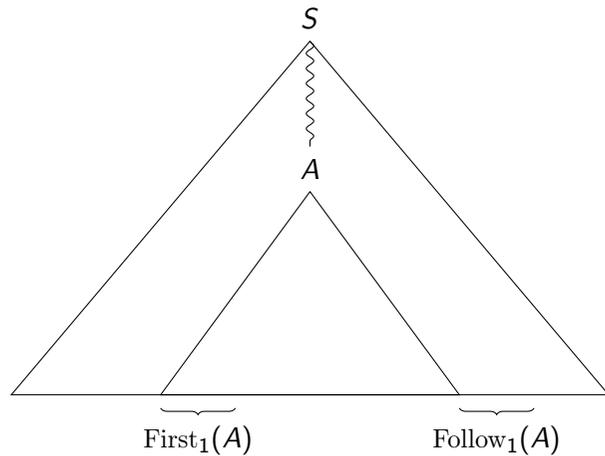
- ▶ Exemple : $E' \rightarrow \epsilon \mid +E$

α	$\text{First}_{\leq 1}(\alpha)$
ϵ	{ ϵ }
+ E	{+}

Si nous voyons + alors on peut utiliser la deuxième alternative pour réécrire E' . Mais quand faut-il appliquer la première ?

- ▶ Il nous manque une information : quels sont les symboles terminaux qui peuvent *suivre* un mot produit par un non-terminal ?

First₁ et Follow₁



La fonction Follow₁

Définition

Soit $G = (\Sigma, N, S, P)$ une grammaire. La fonction $\text{Follow}_1 : N \rightarrow 2^\Sigma$ est définie par

$$\text{Follow}_1(A) = \{c \mid S \rightarrow^* \beta A \gamma, \beta, \gamma \in (N \cup \Sigma)^*, c \in \text{First}_1(\gamma)\}$$

Explication

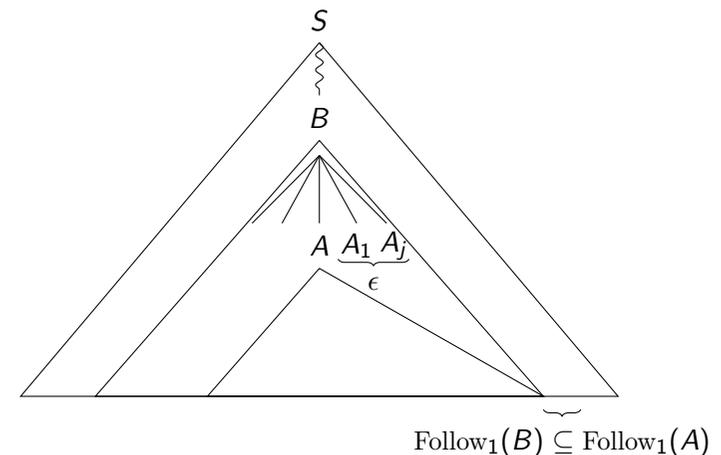
$\text{Follow}_1(A)$ est l'ensemble de tous les symboles terminaux qui peuvent, dans des mots de $\mathcal{L}(G)$, suivre un mot dérivé de A .

Calcul de Follow₁ pour les non-terminaux

- ▶ Grammaire $G = (\Sigma, N, S, P)$.
- ▶ On fait un graphe, avec N comme ensemble de nœuds.
- ▶ On ajoute des symboles de Σ comme valeurs aux nœuds. Initialement, on ajoute à un nœud A , pour toutes les règles $\dots \rightarrow \dots A\alpha$, l'ensemble $\text{First}_1(\alpha)$.
- ▶ On fait une arête de A vers B quand il y a une règle de la forme $A \rightarrow \alpha B B_1 \dots B_n$ où $B_1, \dots, B_n \in \text{EPS}$.
- ▶ Puis on propage les valeurs dans le sens des flèches, jusqu'à ne plus pouvoir propager.

Follow₁ : propagation de gauche à droite

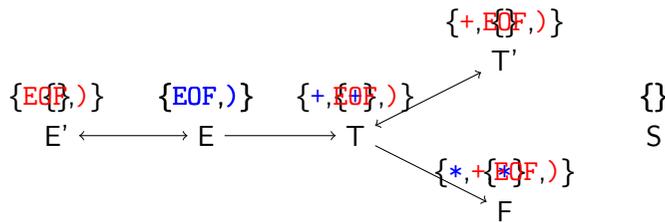
Cas d'une règle $B \rightarrow \dots A A_1 \dots A_j$ avec $A_1, \dots, A_j \in \text{EPS}$



Calcul de Follow₁ sur l'exemple

$$\begin{array}{l} E \rightarrow T E' \quad E' \rightarrow \epsilon \mid +E \quad T \rightarrow F T' \\ T' \rightarrow \epsilon \mid *T \quad F \rightarrow (E) \mid i \quad S \rightarrow E \text{ EOF} \end{array}$$

- ▶ $EPS = \{E', T'\}$ $\text{First}_1(E') = \{+\}$ $\text{First}_1(T') = \{*\}$
- ▶ Initialisation (en bleu)



- ▶ Propagation : ajouter les symboles rouges

Le critère sur l'exemple

Non-terminal	Cas 1	$\text{First}_{\leq 1}$	Cas 2	$\text{First}_{\leq 1}$	Follow_1
E	T E'	{i, (}			{), EOF
E'	ε	{ε}	+ E	{+}	{), EOF
T	F T'	{i, (}			{EOF,), +}
T'	ε	{ε}	* T	{*}	{EOF,), +}
F	(E)	{(}	i	{i}	{EOF,), +, *}
S	E EOF	{i, (}			{}

- ▶ Les $\text{First}_{\leq 1}$ sont disjoints ☺
- ▶ $\text{Follow}_1(E') = \{), \text{EOF}\}$ disjoint avec $\{+\}$ ☺
- ▶ $\text{Follow}_1(T') = \{\text{EOF}, \text{)}, +\}$ disjoint avec $\{*\}$ ☺
- ▶ Conclusion : la grammaire est LL(1)!

Le critère général pour être LL(1)

Théorème

La grammaire $G = (\Sigma, N, S, P)$ est LL(1) ssi pour toutes les alternatives $A \rightarrow \alpha_1 \mid \dots \mid \alpha_n$:

1. $\text{First}_{\leq 1}(\alpha_1), \dots, \text{First}_{\leq 1}(\alpha_n)$ sont disjoints entre eux ;
2. Si $\epsilon \in \text{First}_{\leq 1}(\alpha_i)$, alors pour tous $j \neq i$:

$$\text{First}_{\leq 1}(\alpha_j) \cap \text{Follow}_1(A) = \emptyset$$

Remarque

Condition (1) implique qu'au plus un des ensembles $\text{First}_{\leq 1}(\alpha_i)$ contient ϵ .

Comment choisir la règle dans l'analyse syntaxique

Soit $A \rightarrow \alpha_1 \mid \dots \mid \alpha_n$ une alternative. Il y a deux cas :

1. Soit aucun des $\text{First}_{\leq 1}(\alpha_i)$ ne contient ϵ :
comme avant :
 - ▶ On choisit la règle $A \rightarrow \alpha_i$ quand le symbole suivant est dans $\text{First}_{\leq 1}(\alpha_i)$ (ils sont tous disjoints).
 - ▶ Erreur si aucun tel i existe
2. Soit il existe un (seul) α_i avec $\epsilon \in \text{First}_{\leq 1}(\alpha_i)$:
 - ▶ si le symbole suivant est dans $\text{First}_{\leq 1}(\alpha_j)$: choisir $A \rightarrow \alpha_j$, pour $1 \leq j \leq n$.
 - ▶ si le symbole suivant est dans $\text{Follow}_1(A)$: choisir $A \rightarrow \alpha_i$.
 - ▶ sinon Erreur.

Codage

- ▶ Une seule différence au code montré la semaine dernière :
- ▶ Pour le cas d'une règle $N \rightarrow \alpha_i$ avec $\epsilon \in \text{First}_{\leq 1}(\alpha_i)$: dans la fonction pour le non-terminal N, cas pour le côté droit α_i : on ajoute $\text{Follow}_1(N)$ dans la distinction de cas.
- ▶ Inconvénient : l'arbre de dérivation est tordu car on a transformé la grammaire pour la rendre LL(1).

Fichier parser.ml II

```

| _ -> raise (Error "parsing⊔E")
and parse_Eprime () =
  match lookahead () with
  | Ch ')' | EOF -> (* E' -> epsilon *)
    Node("E'",[Epsilon])
  | Ch '+' -> begin (* E' -> + E *)
    eat (Ch '+');
    let x = parse_E () in
    Node("E'",[Leaf '+';x])
  end
| _ -> raise (Error "parsing⊔E'")
and parse_T () =
  match lookahead () with
  | Ch 'i' | Ch '(' -> begin (* T -> F T' *)
    let x1 = parse_F () in
    let x2 = parse_Tprime () in
    Node("T",[x1;x2])
  end

```

Fichier parser.ml I

```

open Tree
open Reader

exception Error of string
let rec parse_S () =
  match lookahead () with
  | Ch 'i' | Ch '(' -> begin (* S -> E *)
    let x = parse_E () in
    Node("S",[x])
  end
| _ -> raise (Error "parsing⊔S")
and parse_E () =
  match lookahead () with
  | Ch 'i' | Ch '(' -> begin (* E -> T E' *)
    let x1 = parse_T () in
    let x2 = parse_Eprime () in
    Node("E",[x1;x2])
  end

```

Fichier parser.ml III

```

| _ -> raise (Error "parsing⊔T")
and parse_Tprime () =
  match lookahead () with
  | Ch ')' | Ch '+' | EOF -> (* T' -> epsilon *)
    Node("T'",[Epsilon])
  | Ch '*' -> begin (* T' -> * T *)
    eat (Ch '*');
    let x = parse_T () in
    Node("T'",[Leaf '*';x])
  end
| _ -> raise (Error "parsing⊔E'")
and parse_F () =
  match lookahead () with
  | Ch '(' -> begin (* F -> ( E ) *)
    eat (Ch '(');
    let x = parse_E () in
    eat (Ch ')');
    Node("F",[Leaf '(';x;Leaf ')'])
  end

```

Fichier parser.ml IV

```
end
| Ch 'i' -> begin (* F -> i *)
  eat (Ch 'i');
  Node("F",[Leaf 'i'])
end
| _ -> raise (Error "parsing␣F")

let parse () = parse_S ()
```

Grammaires récursives à gauche

- ▶ Une grammaire est *récursive à gauche* quand elle contient une règle

$$N \rightarrow N\alpha$$

- ▶ Exemple :

$$\begin{aligned} S &\rightarrow E \text{ EOF} \\ E &\rightarrow E+T \mid T \\ T &\rightarrow T*F \mid F \\ F &\rightarrow (E) \mid i \end{aligned}$$

- ▶ Parfois on prend une définition plus large : existence d'une dérivation

$$N \rightarrow^+ N\alpha$$

Récursivité à gauche et LL(1)

On va montrer : Si la grammaire G est réduite et récursive à gauche, alors G n'est *pas* LL(1).

Preuve :

- ▶ Puisque G est récursive à gauche, il y a une règle $N \rightarrow N\alpha$.
- ▶ Puisque G est réduite, il y a une autre règle pour N : $N \rightarrow \beta$ (sinon N ne serait pas productif).
- ▶ Deux cas, selon $\beta \not\rightarrow^* \epsilon$ ou $\beta \rightarrow^* \epsilon$.

Cas 1 : $\beta \not\rightarrow^* \epsilon$

On a des règles $N \rightarrow N\alpha$ et $N \rightarrow \beta$, et on a $\beta \not\rightarrow^* \epsilon$.

- ▶ Pour que G soit LL(1), il faut que $\text{First}_{\leq 1}(N\alpha)$ et $\text{First}_{\leq 1}(\beta)$ soient disjoints.
- ▶ On a que $\text{First}_1(\beta) \neq \emptyset$ car G réduit et $\beta \not\rightarrow^* \epsilon$.
- ▶ On a que $\text{First}_1(\beta) \subseteq \text{First}_1(N) \subseteq \text{First}_{\leq 1}(N\alpha)$.
- ▶ On a aussi que $\text{First}_1(\beta) \subseteq \text{First}_{\leq 1}(\beta)$.
- ▶ Donc $\text{First}_{\leq 1}(\beta) \cap \text{First}_{\leq 1}(N\alpha) \neq \emptyset$.

Cas 2 : $\beta \rightarrow^* \epsilon$

On a des règles $N \rightarrow N\alpha$ et $N \rightarrow \beta$; et on a $\beta \rightarrow^* \epsilon$. N est donc annulable.

- ▶ Pour que G soit LL(1), il faut que $\text{First}_{\leq 1}(N\alpha)$ et $\text{Follow}_1(N)$ soient disjoint.
- ▶ On a $\text{First}_1(\alpha) \neq \emptyset$ car sinon $N \rightarrow N\alpha \rightarrow \beta\alpha \rightarrow^* \alpha \rightarrow^* \epsilon$ et aussi $N \rightarrow \beta \rightarrow^* \epsilon$, et la grammaire serait ambiguë.
- ▶ On a que $\text{First}_1(\alpha) \subseteq \text{First}_{\leq 1}(N\alpha)$ car N est annulable.
- ▶ On a aussi que $\text{First}_1(\alpha) \subseteq \text{Follow}_1(N)$.
- ▶ Donc : $\text{Follow}_1(N) \cap \text{First}_{\leq 1}(N\alpha) \neq \emptyset$.