Analyse Syntaxique

Analyse Syntaxique

Introduction

Compilation - Idée

Il s’agit lors de la compilation de

- savoir si un programme est syntaxiquement correct
- construire l’arbre de dérivation qui permettra ensuite de gérer le code cible

Formellement, le programme est un mot sur l’alphabet du langage de programmation, et l’ensemble des programmes syntaxiquement corrects forme un langage

Le langage des programmes syntaxiquement corrects est algébrique, la syntaxe correcte est décrite par une grammaire algébrique

Référence : Dragon book, chapitre 4

Formalisation de l'idée

Si $\mathcal{G} = (\Sigma, V, S, R)$ est la grammaire d'un langage de programmation, un programme candidat est un mot $u \in \Sigma^*$ et savoir si u est syntaxiquement correct revient à tester si $u \in \mathcal{L}(\mathcal{G})$

De manière générale, on sait décider si $u \in \mathcal{L}(\mathcal{G})$ en temps fini

Difficultés à affronter :

- efficacité des algorithmes
- non-déterminisme et ambiguïté éventuels de la grammaire

But de l'algorithme

Considérons le graphe de toutes les dérivations possibles par une grammaire $\mathcal{G} = (\Sigma, V, S, R)$, les nœuds du graphe sont des mots $\alpha \in (V \cup \Sigma)^*$ et il existe un arc de α vers β si $\alpha \to \beta$

Pour un programme $u \in \Sigma^*$, un algorithme d'analyse doit trouver un chemin de S à u dans le graphe (dans la plupart des cas s'il en existe un il en existe plusieurs)

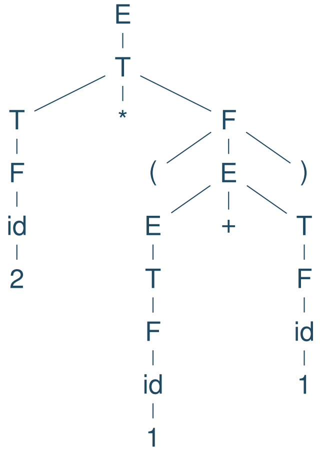
Il est nécessaire également que l'algorithme se souvienne du chemin complet afin de produire un arbre de dérivation

Que cherche-t-on à faire ?

Grammaire des expressions

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid id \\
id & \rightarrow & 1 \mid 2
\end{array}
$$

À analyser : $2 * (1 + 1)$



Types d'analyse

Il existe deux manières d'aborder le problème :

- partir de S et explorer le graphe de proche en proche en essayant de trouver u ➥analyse dite **descendante**
- partir de u et explorer le graphe de proche en proche en orientant les arcs dans le sens inverse jusqu'à retrouver S ➥analyse dite **ascendante**

Moyen mnémotechique

En informatique les arbres poussent la racine en haut et les feuilles en bas, l'analyse de S vers u est donc descendante, celle de u vers S ascendante

Analyse Syntaxique

Analyse descendante (LL)

Principe de l'analyse LL (1/4)

LL signifie : left-to-right parse, leftmost derivation

L'analyse s'appuie sur un automate à pile qui cherche à construire une dérivation gauche en plaçant sur la pile les symboles qui doivent encore être ré-écrits

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire algébrique, on construit un automate à pile à acceptation par pile vide : $\mathcal{A} = (\Sigma, V \cup \Sigma, S, Q, q_0, F, \delta)$ tel que

$$\begin{array}{rl} \delta = & \{(q_0, z) \xrightarrow{\varepsilon} (q_0, z') \mid z \to z' \in R\} \\ \cup & \{(q_0, x) \xrightarrow{x} (q_0, \varepsilon) \mid x \in \Sigma\} \end{array}$$

➥ Problème : cette technique conduit en général à un automate non-déterministe

Principe de l'analyse LL (2/4)

Le non-déterminisme vient du choix possible pour dériver un non terminal *T* dès lors qu'il existe plusieurs α tels que $T \rightarrow \alpha$:

- si α commence par un terminal, la décision peut-être prise en comparant le terminal avec l'entrée
- si α commence par un non terminal, la décision peut être plus complexe

Pour lever le non-déterminisme, on va autoriser l'automate à « explorer la suite du mot », en regardant les *k* prochaines lettres de l'entrée

Principe de l'analyse LL (3/4)

On souhaite une analyse déterministe

➥À chaque pas de calcul, l'automate décide de façon déterministe quelle règle appliquer en fonction des k prochains symboles d'entrée et du sommet de pile

Le choix n'est jamais remis en question :

- si l'analyse conclut sur un succès le mot appartient au langage
- en cas d'échec le mot n'appartient pas au langage *i.e.* il n'y a pas d'autre essai à faire

Le nombre k est fixé pour une analyse donnée, on dira qu'une grammaire/un langage est $LL(k)$

Principe de l'analyse LL (4/4)

L'idée va être de construire une **table de prédiction** à partir de la grammaire qui :

- étant donné un non-terminal à dériver et les k prochaines lettres de l'entrée
- indique les parties droites de règles susceptibles de mener à la lettre d'entrée à reconnaître

Alors :

- s'il y a au plus une dérivation possible dans chaque case, la grammaire est dite LL(k)

Les grammaires $LL(k)$ permettent de construire des analyseurs déterministes de gauche à droite (left-to-right) construisant les dérivations gauches (leftmost derivation) avec un regard en avant (look-ahead) de k symboles d'entrée

Comment ça marche - Exemple 1

Considérons la grammaire

$$\begin{array}{lll} S & \rightarrow & aSb \mid cT \\ T & \rightarrow & aT \mid b \end{array}$$

et sa table de prédiction

	a	b	c
S	aSb		cT
T	aT	b	

Mot à analyser

$$\begin{array}{ccccccc} a & a & c & a & b & b & b \\ \uparrow & & & & & & \end{array}$$

Rappel de la fonction de transition

$$\begin{array}{lll} \delta = & & \{(q_0, z) \xrightarrow{\varepsilon} (q_0, z') \mid z \rightarrow z' \in R\} \\ & \cup & \{(q_0, x) \xrightarrow{x} (q_0, \varepsilon) \mid x \in \Sigma\} \end{array}$$

			a		c		a					
	a		S	S	T	T	T	T	b			
	S	S	b	b	b	b	b	b	b	b		
S	b	b	b	b	b	b	b	b	b	b	b	

Comment ça marche ? - Exemple 2 (1/2)

Considérons la grammaire

$$\begin{array}{lcl} S & \rightarrow & aSb \mid cT | aU \\ T & \rightarrow & aT \mid c \\ U & \rightarrow & dU \mid d \end{array}$$

et sa table de prédiction à 1 lettre

	a	*b*	*c*	*d*
S	*aSb* *aU*		*cT*	
T	*aT*		*c*	
U				*dU* *d*

Table de prédiction avec un look-ahead de 2

	aa	*ac*	*ad*	*ca*	*cb*	*cc*	*db*	*dd*	*c*	*d*
S	*aSb*	*aSb*	*aU*	*cT*		*cT*				
T	*aT*	*aT*			*c*				*c*	
U							*d*	*dU*		*d*

➥ La grammaire est $LL(2)$

Comment ça marche ? - Exemple 2 (2/2)

Table de prédiction avec un look-ahead de 2

	aa	ac	ad	ca	cb	cc	db	dd	c	d
S	aSb	aSb	aU	cT		cT				
T	aT	aT			c				c	
U							d	dU		d

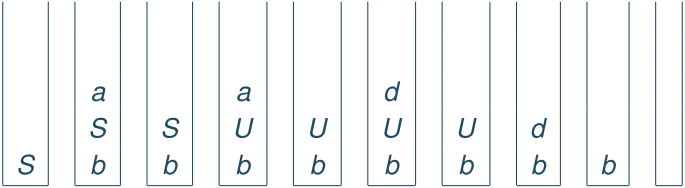
Mot à analyser

$a \quad a \quad d \quad d \quad b$

$\uparrow$

Rappel de la fonction de transition

$$\begin{aligned} \delta = \quad & \{(q_0, z) \xrightarrow{\varepsilon} (q_0, z') \mid z \to z' \in R\} \\ \cup \quad & \{(q_0, x) \xrightarrow{x} (q_0, \varepsilon) \mid x \in \Sigma\} \end{aligned}$$



Construction de la table de prédiction

- Les tables des exemples précédents sont construites « à la main »
- La construction est assez « facile » lorsque k est petit, que les règles sont peu nombreuses et commencent, comme dans les exemples par des terminaux

➥ Ce n'est évidemment pas le cas général

L'idée est que si des règles $T \to U\alpha$ et $U \to a\beta$ appartiennent à la grammaire, alors T peut être considérée lorsque la prochaine règle de l'entrée est a

➥ Ce qui va mener à une méthode pour construire les tables

Cette construction se base sur la construction de 2 ensembles : *First* et *Follow* pour $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire algébrique

Analyse Syntaxique

Analyse $LL(1)$

Calcul de *First* - Intuition

Intuitivement, *First* va permettre de calculer pour un mot de $(\Sigma \cup V)^*$ (aussi appelé « proto-mot ») les premières lettres des mots qui en seront dérivés

Par exemple pour une règle $\alpha \to aT \mid T_1 T_2 \dots T_n$:

- avec aT on sait que l'on peut réécrire α en un mot qui commence par a
- pour $\alpha \to T_1 T_2 \dots T_n$:
  - on peut réécrire α en un mot qui commence par les premières lettres des mots que l'on peut réécrire à partir de T_1
  - si T_1 peut se réécrire en le mot vide, alors on peut réécrire α en un mot qui commence par les premières lettres des mots que l'on peut réécrire à partir de T_2
  - si $T_1 T_2$ peut se réécrire en le mot vide, alors on peut réécrire α en un mot qui commence par les premières lettres des mots que l'on peut réécrire à partir de T_3
  - ...

Définition de *First*

Définition

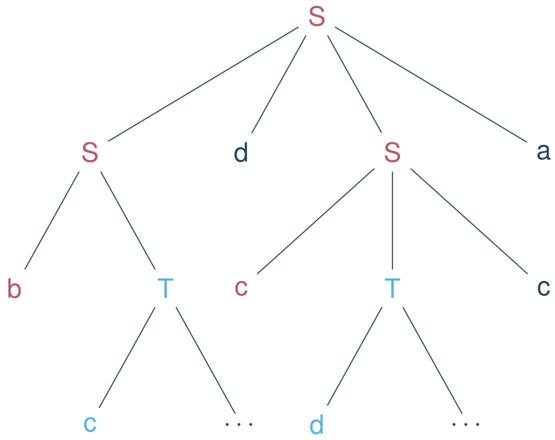
$$\begin{array}{rrcl} \mathit{First} : & (\Sigma \cup V)^+ & \to & 2^{\Sigma \cup \{\varepsilon\}} \\ & \alpha & \mapsto & \{x \in \Sigma \mid \alpha \underset{*}{\to} xu \in \Sigma^+\} \\ & & & \cup\{\varepsilon \text{ si } \alpha \underset{*}{\to} \varepsilon\} \end{array}$$

Calcul

- $\forall x \in \Sigma, \mathit{First}(x) = \{x\}$
- $\forall T \in V, \mathit{First}(T) = \bigcup_{T \to \alpha_1 \ldots \alpha_n \in R} \bigcup_{\alpha_1 \ldots \alpha_{i-1} \underset{*}{\to} \varepsilon} \mathit{First}(\alpha_i) \cup \{\varepsilon \text{ si } T \underset{*}{\to} \varepsilon\}$

➥Il est suffisant de calculer les *First* pour les non terminaux

Calcul de *First* - Idée sur les arbres de dérivation



Calcul de *First* - Exemple

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TU \mid Xc \\ T & \rightarrow & aUd \mid \varepsilon \\ U & \rightarrow & bX \mid \varepsilon \\ X & \rightarrow & dX \mid e \end{array}$$

Calculons *First* pour les non-terminaux

- $First(S) = First(T) \cup First(X) \cup First(U) \cup \{\varepsilon\}$
- $First(T) = First(a) \cup \{\varepsilon\}$
- $First(U) = First(b) \cup \{\varepsilon\}$
- $First(X) = First(d) \cup First(e)$

Calcul de *First* - Exemple

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TU \mid Xc \\ T & \rightarrow & aUd \mid \varepsilon \\ U & \rightarrow & bX \mid \varepsilon \\ X & \rightarrow & dX \mid e \end{array}$$

Calculons *First* pour les non-terminaux

- $First(S) = First(T) \cup First(X) \cup First(U) \cup \{\varepsilon\}$
- $First(T) = First(a) \cup \{\varepsilon\} = \{a, \varepsilon\}$
- $First(U) = First(b) \cup \{\varepsilon\} = \{b, \varepsilon\}$
- $First(X) = First(d) \cup First(e) = \{d, e\}$

Calcul de *First* - Exemple

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TU \mid Xc \\ T & \rightarrow & aUd \mid \varepsilon \\ U & \rightarrow & bX \mid \varepsilon \\ X & \rightarrow & dX \mid e \end{array}$$

Calculons *First* pour les non-terminaux

- $First(S) = First(T) \cup First(X) \cup First(U) \cup \{\varepsilon\} = \{a, b, d, e, \varepsilon\}$
- $First(T) = First(a) \cup \{\varepsilon\} = \{a, \varepsilon\}$
- $First(U) = First(b) \cup \{\varepsilon\} = \{b, \varepsilon\}$
- $First(X) = First(d) \cup First(e) = \{d, e\}$

Calcul de *First* pour un symbole

- Pour chaque $x \in \Sigma$, $\mathit{First}(x) = \{x\}$

Ensuite par calcul de point fixe, itérer jusqu'à ce qu'aucun ensemble *First* n'évolue pour chaque $T \in V$

1. Pour chaque règle $T \to T_1 \ldots T_n$
   1.1 Ajouter $\mathit{First}(T_1) \setminus \varepsilon$ à $\mathit{First}(T)$
   1.2 $i = 1$
   1.3 Tant que $i < n$ et $\varepsilon \in \mathit{First}(T_i)$: ajouter $\mathit{First}(T_{1+1}) \setminus \varepsilon$ à $\mathit{First}(T)$; incrémenter i
   1.4 Si $\varepsilon \in \mathit{First}(T_1), \ldots, \varepsilon \in \mathit{First}(T_n)$ ajouter ε à $\mathit{First}(T)$
2. si $T \to \varepsilon \in R$ ajouter ε à $\mathit{First}(T)$

Calcul de *First* - Exemple 1

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & aSb \mid cT \\ T & \rightarrow & aT \mid b \end{array}$$

Alors $First(a) = \{a\}$, $First(b) = \{b\}$, calculons *First* pour les non-terminaux

- Itération 1 : $First(S) = \{a, c\}$, $First(T) = \{a, b\}$
- Itération 2 : pas de changement

nous avons donc $First(S) = \{a, c\}$, $First(T) = \{a, b\}$

Calcul de *First* - Exemple 2

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TU \mid Xc \\ T & \rightarrow & aUd \mid \varepsilon \\ U & \rightarrow & bX \mid \varepsilon \\ X & \rightarrow & dX \mid e \end{array}$$

Déroulons l'algorithme de calcul de *First* pour les non-terminaux

	S	T	U	X
1.1 1.3 2	$\cup First(T) \cup First(X)$ $\varepsilon \in First(T) \Rightarrow \cup First(U)$	$\cup First(a)$ $\cup\{\varepsilon\}$	$\cup First(b)$ $\cup\{\varepsilon\}$	$\cup First(d) \cup First(e)$
It 1	$\emptyset$	$\{a, \varepsilon\}$	$\{b, \varepsilon\}$	$\{d, e\}$
It 2	$\{a, \varepsilon, d, e, b\}$	$\{a, \varepsilon\}$	$\{b, \varepsilon\}$	$\{d, e\}$
It 3	$\{a, \varepsilon, d, e, b\}$	$\{a, \varepsilon\}$	$\{b, \varepsilon\}$	$\{d, e\}$

Calcul de *First* - Pour un mot

À partir du calcul de *First*(α) pour $\alpha \in \Sigma \cup V$, on peut calculer *First*(α) pour $\alpha \in (\Sigma \cup V)^*$, en prenant les mêmes précautions avec les ε que durant le calcul précédent

Pour tout $\alpha = \alpha_1 \dots \alpha_n$, *First*$(\alpha)$ est calculé de la manière suivante :

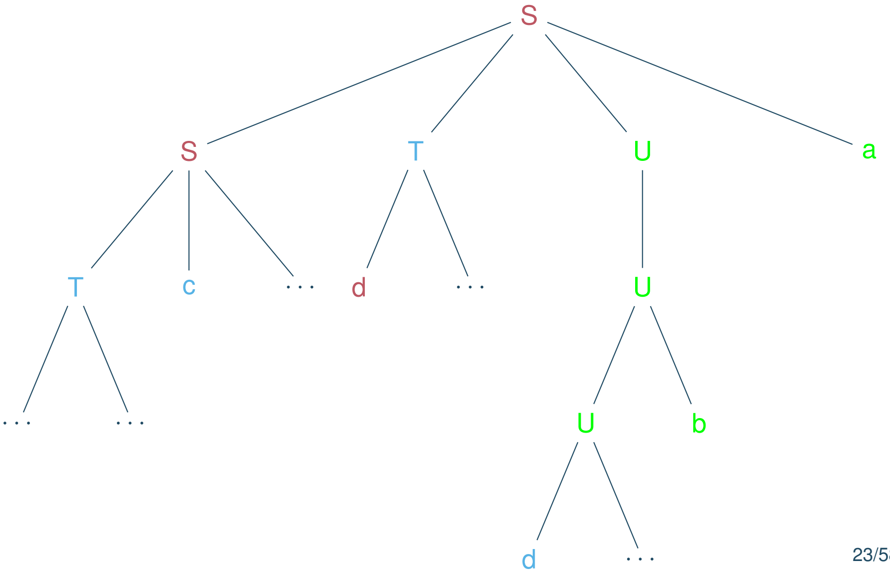
- Ajouter *First*$(\alpha_1) \setminus \varepsilon$ à *First*(α)
- $i = 1$
- Tant que $i < n$ et $\varepsilon \in$ *First*(α_i) : ajouter *First*$(\alpha_{i+1}) \setminus \varepsilon$ à *First*(α) ; incrémenter i
- Si $\varepsilon \in$ *First*$(\alpha_1), \dots, \varepsilon \in$ *First*(α_n) ajouter ε à *First*(α)

Calcul de *Follow* - Intuition

La fonction *Follow* est définie pour les non terminaux, elle calcule le premier symbole terminal qui peut suivre le mot produit par le non terminal en question

- Par convention, on note $\#$ la fin du mot, ce qui permet de considérer que la fin de mot peut suivre un non terminal, en particulier $\# \in \mathit{Follow}(S)$
- Attention, $\mathit{Follow}(T)$ va être calculé à partir des règles dans lesquelles T apparaît en partie droite de règle
- Attention, si T apparaît plusieurs fois en partie droite d'une règle il faut considérer toutes les occurences

Calcul de *Follow* - Idée sur les arbres de dérivation



Calcul de *Follow*

<u>Définition</u>

$$
\begin{array}{rlcl}
\textit{Follow} : & V & \to & 2^{\Sigma \cup \{\#\}} \\
& T & \mapsto & \{x \in \Sigma \mid S \underset{*}{\to} \alpha T x \beta\} \cup \{\# \text{ si } S \underset{*}{\to} \alpha T\}
\end{array}
$$

ou encore

$$
T \mapsto \{x \in \mathit{First}(\beta) \setminus \{\varepsilon\} \mid S \underset{*}{\to} \alpha T \beta\} \cup \{\# \text{ si } S \underset{*}{\to} \alpha T\}
$$

<u>Grammaire $LL(1)$</u>

Une grammaire algébrique $\mathcal{G} = (\Sigma, V, S, R)$ est $LL(1)$ si pour tout $T \in V$, pour toutes les règles $T \to \alpha_i \in R$ avec $0 \leq i \leq n$ alors :

- $\mathit{First}(\alpha_i) \cap \mathit{First}(\alpha_j) = \emptyset$ si $i \neq j$
- si $T \to \varepsilon$ alors $\forall 0 \leq i \leq n, \mathit{First}(\alpha_i) \cap \mathit{Follow}(T) = \emptyset$

Calcul de *Follow*

Calcul de *Follow* pour les non-terminaux :

- Ajouter $\#$ à $\mathit{Follow}(S)$

Ensuite itérer jusqu'à ce qu'aucun ensemble *Follow* n'évolue entre 2 itérations

1. Pour chaque règle $T \rightarrow T_1 \ldots T_n$
   1.1 Pour tout $1 \leq i < n$ ajouter $\mathit{First}(T_{i+1}) \setminus \{\varepsilon\}$ à $\mathit{Follow}(T_i)$
   1.2 Ajouter $\mathit{Follow}(T)$ à $\mathit{Follow}(T_n)$
   1.3 $i = n - 1$
   1.4 Tant que $i > 0$ et que $\varepsilon \in \mathit{First}(T_{i+1})$ ajouter $\mathit{Follow}(T)$ à $\mathit{Follow}(T_i)$

Calcul de *Follow* - Exemple

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TU \mid Xc \\ T & \rightarrow & aUd \mid \varepsilon \\ U & \rightarrow & bX \mid \varepsilon \\ X & \rightarrow & dX \mid e \end{array}$$

Rappel des *First*

- $First(S) = \{a, b, d, e, \varepsilon\}$
- $First(T) = \{a, \varepsilon\}$
- $First(U) = \{b, \varepsilon\}$
- $First(X) = \{d, e\}$

Déroulons l'algorithme de calcul de *Follow* pour les non-terminaux

Étapes	S	T	U	X
1.1	$Follow(S) = \{\#\}$	$\cup First(U)$	$\cup First(d)$	$\cup First(c)$
1.2			$\cup Follow(S)$	$\cup Follow(U)$
1.4		$\cup Follow(S)$		
Itération 1	$\{\#\}$	$\{\#\}$	$\{d, \#\}$	$\{c\}$
Itération 2	$\{\#\}$	$\{\#, b\}$	$\{d, \#\}$	$\{c, d, \#\}$
Itération 3	$\{\#\}$	$\{\#, b\}$	$\{d, \#\}$	$\{c, d, \#\}$

Calcul de la table de prédiction $LL(1)$

Après avoir calculé :

- les ensembles $\mathit{First}(\alpha)$ pour tout $\alpha \in \Sigma \cup V$
- les ensembles $\mathit{Follow}(T)$ pour tout $T \in V$

Calculer la table M

- Pour chaque règle $T \to \alpha$
  - Pour chaque $a \in \mathit{First}(\alpha)$, ajouter α à la case $M[T, a]$
  - Si $\varepsilon \in \mathit{First}(\alpha)$, ajouter α à la case $M[T, b]$ pour tout $b \in \mathit{Follow}(T)$
  - Si $\varepsilon \in \mathit{First}(\alpha)$ et $\# \in \mathit{Follow}(T)$, ajouter α à la case $M[T, \#]$
- Les cases vides indiquent les cas d'erreur

⚠ Ici on considère $\mathit{First}(\varepsilon) = \{\varepsilon\}$

Calcul de la table de prédiction $LL(1)$ - Exemple

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TU \mid Xc \\ T & \rightarrow & aUd \mid \varepsilon \\ U & \rightarrow & bX \mid \varepsilon \\ X & \rightarrow & dX \mid e \end{array}$$

Rappel des *First*

- $First(S) = \{a, b, d, e, \varepsilon\}$
- $First(T) = \{a, \varepsilon\}$
- $First(U) = \{b, \varepsilon\}$
- $First(X) = \{d, e\}$

Rappel des *Follow*

- $Follow(S) = \{\#\}$
- $Follow(T) = \{\#, b\}$
- $Follow(U) = \{\#, d\}$
- $Follow(X) = \{d, c, \#\}$

	a	b	c	d	e	$\#$
S	TU	TU				TU
T						
U						
X						

- $First(TU) = \{a, b, \varepsilon\}$
- $First(Xc) = \{d, e\}$
- $First(aUd) = \{a\}$
- $First(bX) = \{b\}$
- $First(dX) = \{d\}$

Calcul de la table de prédiction $LL(1)$ - Exemple

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TU \mid Xc \\ T & \rightarrow & aUd \mid \varepsilon \\ U & \rightarrow & bX \mid \varepsilon \\ X & \rightarrow & dX \mid e \end{array}$$

Rappel des *First*

- $First(S) = \{a, b, d, e, \varepsilon\}$
- $First(T) = \{a, \varepsilon\}$
- $First(U) = \{b, \varepsilon\}$
- $First(X) = \{d, e\}$

Rappel des *Follow*

- $Follow(S) = \{\#\}$
- $Follow(T) = \{\#, b\}$
- $Follow(U) = \{\#, d\}$
- $Follow(X) = \{d, c, \#\}$

	a	b	c	d	e	$\#$
S	TU	TU		Xc	Xc	TU
T						
U						
X						

- $First(TU) = \{a, b, \varepsilon\}$
- $First(Xc) = \{d, e\}$
- $First(aUd) = \{a\}$
- $First(bX) = \{b\}$
- $First(dX) = \{d\}$

Calcul de la table de prédiction $LL(1)$ - Exemple

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TU \mid Xc \\ T & \rightarrow & aUd \mid \varepsilon \\ U & \rightarrow & bX \mid \varepsilon \\ X & \rightarrow & dX \mid e \end{array}$$

Rappel des *First*

- $First(S) = \{a, b, d, e, \varepsilon\}$
- $First(T) = \{a, \varepsilon\}$
- $First(U) = \{b, \varepsilon\}$
- $First(X) = \{d, e\}$

Rappel des *Follow*

- $Follow(S) = \{\#\}$
- $Follow(T) = \{\#, b\}$
- $Follow(U) = \{\#, d\}$
- $Follow(X) = \{d, c, \#\}$

	a	b	c	d	e	$\#$
S	TU	TU		Xc	Xc	TU
T	aUd					
U						
X						

- $First(TU) = \{a, b, \varepsilon\}$
- $First(Xc) = \{d, e\}$
- $First(aUd) = \{a\}$
- $First(bX) = \{b\}$
- $First(dX) = \{d\}$

Calcul de la table de prédiction $LL(1)$ - Exemple

Exemple de grammaire

$$
\begin{array}{lcl}
S & \rightarrow & TU \mid Xc \\
T & \rightarrow & aUd \mid \varepsilon \\
U & \rightarrow & bX \mid \varepsilon \\
X & \rightarrow & dX \mid e
\end{array}
$$

Rappel des *First*

- $First(S) = \{a, b, d, e, \varepsilon\}$
- $First(T) = \{a, \varepsilon\}$
- $First(U) = \{b, \varepsilon\}$
- $First(X) = \{d, e\}$

Rappel des *Follow*

- $Follow(S) = \{\#\}$
- $Follow(T) = \{\#, b\}$
- $Follow(U) = \{\#, d\}$
- $Follow(X) = \{d, c, \#\}$

	a	b	c	d	e	$\#$
S	TU	TU		Xc	Xc	TU
T	aUd	ε				ε
U						
X						

- $First(TU) = \{a, b, \varepsilon\}$
- $First(Xc) = \{d, e\}$
- $First(aUd) = \{a\}$
- $First(bX) = \{b\}$
- $First(dX) = \{d\}$

Calcul de la table de prédiction $LL(1)$ - Exemple

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TU \mid Xc \\ T & \rightarrow & aUd \mid \varepsilon \\ U & \rightarrow & bX \mid \varepsilon \\ X & \rightarrow & dX \mid e \end{array}$$

Rappel des *First*

- $First(S) = \{a, b, d, e, \varepsilon\}$
- $First(T) = \{a, \varepsilon\}$
- $First(U) = \{b, \varepsilon\}$
- $First(X) = \{d, e\}$

Rappel des *Follow*

- $Follow(S) = \{\#\}$
- $Follow(T) = \{\#, b\}$
- $Follow(U) = \{\#, d\}$
- $Follow(X) = \{d, c, \#\}$

	a	b	c	d	e	$\#$
S	TU	TU		Xc	Xc	TU
T	aUd	ε				ε
U		bX				
X						

- $First(TU) = \{a, b, \varepsilon\}$
- $First(Xc) = \{d, e\}$
- $First(aUd) = \{a\}$
- $First(bX) = \{b\}$
- $First(dX) = \{d\}$

Calcul de la table de prédiction $LL(1)$ - Exemple

Exemple de grammaire

$$
\begin{array}{lcl}
S & \rightarrow & TU \mid Xc \\
T & \rightarrow & aUd \mid \varepsilon \\
U & \rightarrow & bX \mid \varepsilon \\
X & \rightarrow & dX \mid e
\end{array}
$$

Rappel des *First*

- $First(S) = \{a, b, d, e, \varepsilon\}$
- $First(T) = \{a, \varepsilon\}$
- $First(U) = \{b, \varepsilon\}$
- $First(X) = \{d, e\}$

Rappel des *Follow*

- $Follow(S) = \{\#\}$
- $Follow(T) = \{\#, b\}$
- $Follow(U) = \{\#, d\}$
- $Follow(X) = \{d, c, \#\}$

	a	b	c	d	e	$\#$
S	TU	TU		Xc	Xc	TU
T	aUd	ε				ε
U		bX		ε		ε
X						

- $First(TU) = \{a, b, \varepsilon\}$
- $First(Xc) = \{d, e\}$
- $First(aUd) = \{a\}$
- $First(bX) = \{b\}$
- $First(dX) = \{d\}$

Calcul de la table de prédiction $LL(1)$ - Exemple

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TU \mid Xc \\ T & \rightarrow & aUd \mid \varepsilon \\ U & \rightarrow & bX \mid \varepsilon \\ X & \rightarrow & dX \mid e \end{array}$$

Rappel des *First*

- $First(S) = \{a, b, d, e, \varepsilon\}$
- $First(T) = \{a, \varepsilon\}$
- $First(U) = \{b, \varepsilon\}$
- $First(X) = \{d, e\}$

Rappel des *Follow*

- $Follow(S) = \{\#\}$
- $Follow(T) = \{\#, b\}$
- $Follow(U) = \{\#, d\}$
- $Follow(X) = \{d, c, \#\}$

	a	b	c	d	e	$\#$
S	TU	TU		Xc	Xc	TU
T	aUd	ε				ε
U		bX		ε		ε
X				dX		

- $First(TU) = \{a, b, \varepsilon\}$
- $First(Xc) = \{d, e\}$
- $First(aUd) = \{a\}$
- $First(bX) = \{b\}$
- $First(dX) = \{d\}$

Calcul de la table de prédiction $LL(1)$ - Exemple

Exemple de grammaire

$$\begin{aligned} S &\rightarrow TU \mid Xc \\ T &\rightarrow aUd \mid \varepsilon \\ U &\rightarrow bX \mid \varepsilon \\ X &\rightarrow dX \mid e \end{aligned}$$

Rappel des *First*

- $First(S) = \{a, b, d, e, \varepsilon\}$
- $First(T) = \{a, \varepsilon\}$
- $First(U) = \{b, \varepsilon\}$
- $First(X) = \{d, e\}$

Rappel des *Follow*

- $Follow(S) = \{\#\}$
- $Follow(T) = \{\#, b\}$
- $Follow(U) = \{\#, d\}$
- $Follow(X) = \{d, c, \#\}$

	a	b	c	d	e	$\#$
S	TU	TU		Xc	Xc	TU
T	aUd	ε				ε
U		bX		ε		ε
X				dX	e	

- $First(TU) = \{a, b, \varepsilon\}$
- $First(Xc) = \{d, e\}$
- $First(aUd) = \{a\}$
- $First(bX) = \{b\}$
- $First(dX) = \{d\}$

Après avoir vu l'analyse $LL(1)$

L'analyse $LL(1)$ est utile et relativement facile à mettre en œuvre, malheureusement toutes les grammaires ne sont pas $LL(1)$, il est possible de les transformer pour se rapprocher d'une forme intéressante par :

- factorisation gauche de la grammaire
- élimination des récursions gauches (très embarassantes pour les analyses descendantes récursives, boucle infinie)

Analyse Syntaxique

Transformations de grammaires

Grammaire $SLL(1)$

Certaines grammaires ont une forme particulière qui se prête idéalement à l'analyse $LL(1)$

Une grammaire algébrique $\mathcal{G} = (\Sigma, V, S, R)$ est dite **simple** $LL(1)$ **ou** $SLL(1)$ si et seulement si

- toute règle $T \rightarrow \alpha \in R$ est telle que $\alpha = x\alpha'$ avec $x \in \Sigma$
- toute paire de règles $T \rightarrow x_1\alpha_1 \in R$ et $T \rightarrow x_2\alpha_2 \in R$ avec $x_1, x_2 \in \Sigma$ est telle que $x_1 \neq x_2$ ou $\alpha_1 = \alpha_2$

➥ La forme de la grammaire rend la construction de l'analyseur immédiate

➥ Malheureusement, c'est une classe restreinte des grammaires algébriques, on peut néanmoins essayer de s'en rapprocher

Factorisation gauche d'une grammaire

Une grammaire algébrique $\mathcal{G} = (\Sigma, V, S, R)$ peut contenir des règles de dérivations pour la même variable commençant par la même chaîne

➥Une analyse $LL(1)$ ne pourra déterminer quelle règle choisir

On applique donc une transformation de la grammaire pour éliminer ce problème en itérant, tant qu'il existe deux règles permettant de dériver le même terminal ayant un préfixe commun

- Pour chaque non-terminal $T \in V$ tel que $T \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$
- Calculer le plus grand préfixe commun α pour des α_i
- Ajouter une nouvelle variable T_α
- Remplacer la règle $T \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$ par les règles
    - $T \to \alpha T_\alpha$
    - $\{T \to \alpha_i \mid \alpha^{-1}\alpha_i = \alpha_i\}$
- Ajouter les nouvelles règles
    - $\{T_\alpha \to \alpha^{-1}\alpha_i \mid \alpha^{-1}\alpha_i \neq \alpha_i\}$

Factorisation gauche d'une grammaire - Exemple

Par exemple pour une instruction de « choix » la grammaire suivante

$$\begin{array}{lcl} stm & \rightarrow & \textbf{if } E \textbf{ then } stm \textbf{ else } stm \\ & | & \textbf{if } E \textbf{ then } stm \\ E & \rightarrow & b \end{array}$$

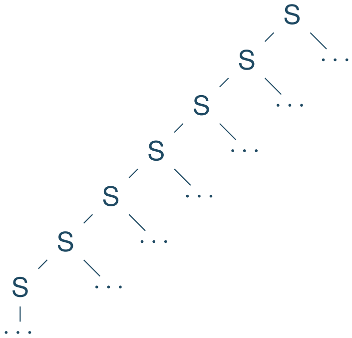
très simple à lire par un humain, ne permet pas de choisir aisément la règle à appliquer arpès avoir lu un **if**

Factorisation Le plus long préfixe commun des deux règles est **if** E **then** stm . On introduit une nouvelle variable stm_{suite}, qui va permettre de retarder le moment du choix de la règle à appliquer

$$\begin{array}{lcl} stm & \rightarrow & \textbf{if } E \textbf{ then } stm\ stm_{suite} \\ stm_{suite} & \rightarrow & \textbf{else } stm \mid \varepsilon \\ E & \rightarrow & b \end{array}$$

Récursivité gauche d'une grammaire

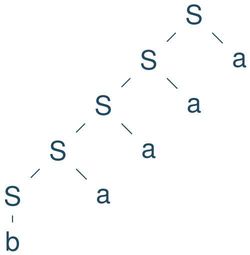
Si un algorithme réalise une exploration « en profondeur d'abord » et que la grammaire est récursive gauche, il peut boucler



Récursivité gauche immédiate

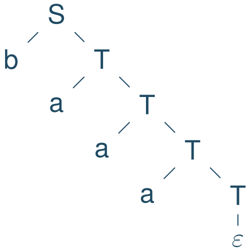
Une variable T de la grammaire $\mathcal{G} = (\Sigma, V, S, R)$ est dite **récursive à gauche immédiate** si il existe $T \rightarrow T\alpha \in R$

Exemple pour ba^* $\quad S \rightarrow Sa \mid b$



On ajoute une nouvelle variable

$$\begin{aligned} S &\rightarrow bT \\ T &\rightarrow aT \mid \varepsilon \end{aligned}$$



Supprimer la récursivité gauche immédiate

Pour toute variable $T \in V$ telle que

$$T \to T\alpha_1 \mid \dots T\alpha_n \mid \beta_1 \mid \dots \beta_m$$

avec $\beta_1 \neq \varepsilon, \dots \beta_m \neq \varepsilon$ et aucun des $\beta_1, \dots, \beta_m$ ne débutant par T, on remplace la règle par

$$\begin{array}{lcl} T & \to & \beta_1 U \mid \dots \beta_m U \\ U & \to & \alpha_1 U \mid \dots \alpha_n U \mid \varepsilon \end{array}$$

Exemple : les expressions

$$\begin{array}{lcl} E & \to & E + T \mid T \\ T & \to & T * F \mid F \\ F & \to & (E) \mid id \end{array}$$

$$\begin{array}{lcl} E & \to & TE' \\ E' & \to & +TE' \mid \varepsilon \\ T & \to & FT' \\ T' & \to & *FT' \mid \varepsilon \\ F & \to & (E) \mid id \end{array}$$

Supprimer la récursivité gauche

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire algébrique sans ε-règles et sans cycle

- Choisir un ordre pour les variables, notons le $T_1, \ldots, T_n$
- Pour chaque variable T_i, i variant de 1 à n
  - Pour chaque variable T_j, j variant de 1 à $i-1$
    - remplacer chaque règle $T_i \to T_j\alpha$ par $T_i \to \beta_1\alpha \mid \cdots \mid \beta_k\alpha$ où $T_j \to \beta_1 \mid \cdots \mid \beta_k$ sont les règles de dérivations courantes pour T_j
  - éliminer la récursité gauche immédiate pour les T_i

Après une itération i, les règles de type $T_k \to T_l\gamma$ avec $k \leq i$ restant sont telles que $l > k$

Forme normale de Greibach

Nous avons vu la forme normale de Chomsky dans la partie sur les grammaires, il existe une autre forme normale qui généralise en quelque sorte la suppression des réccursions gauches

Une grammaire algébrique $\mathcal{G} = (\Sigma, V, S, R)$ est dite **en forme normale de Greibach (GNF)** si pour toute $T \to \alpha \in R$, $\alpha \in \Sigma V^*$

Remarque

On peut considérer les langages contenant le mot vide en ajoutant une règle permettant uniquement de dériver l'axiome non récursif en le mot vide

Mise en forme normale de Greibach (1/3)

Référence : Livre d'Olivier Carton *Langages formels, calculabilité et complexité*

Soit une grammaire algébrique $\mathcal{G}_0 = (\Sigma, V, X_1, R)$ avec $V = \{X_1, \ldots, X_n\}$, la construction de la mise en forme normale de Greibach de $\mathcal{G}_0$ est itérative, on construit successivement des $\mathcal{G}_i = (\Sigma, V_i, X_1, R_i)$

Construction

Pour chaque grammaire $\mathcal{G}_i$, les variables $X_1, \ldots, X_i$ n'apparaissent pas en tête de partie droite de règle

Supposons que l'on ait construit la suite de grammaires jusqu'à $\mathcal{G}_{i-1}$ en préservant la propriété construisons $\mathcal{G}_i$

Mise en forme normale de Greibach (2/3)

Il faut pour cela traiter le cas de X_i, nous avons :

$$X_i \to X_i\alpha_1 + \cdots + X_i\alpha_k + \beta_1 + \cdots + \beta_l$$

avec les β_j qui ne commencent pas par X_i

On peut alors supprimer la récursivité gauche de la dérivation X_i en introduisant une nouvelle variable X_i' et en remplaçant les règles ci-dessus par

$$\begin{array}{rcl} X_i & \to & \beta_1 X_i' + \cdots + \beta_l X_i' + \beta_1 + \cdots + \beta_l \\ X_i' & \to & \alpha_1 X_i' + \cdots + \alpha_k X_i' + \alpha_1 + \cdots + \alpha_k \end{array}$$

Exemple pour se convaincre

$$\begin{array}{rcl} (old)X & \to & Xa + Xb + c + d \\ (new)X & \to & cX' + dX' + c + d \\ X' & \to & aX' + bX' + a + b \end{array}$$

Mise en forme normale de Greibach (3/3)

La transformation de X_i peut introduire de nouveaux problèmes :

- les α_p peuvent commencer par des X_j avec $0 < j \leq i$
- X_i peut être la première lettre de parties droites d'autres règles

Chaque règle $T \to X_j\alpha$ de R_{i-1} ou nouvellement produite, avec $0 < j \leq i$ et $X_j \to \gamma_1 + \cdots + \gamma_m$ est remplacée par

$$T \to \gamma_1\alpha + \cdots + \gamma_m\alpha$$

➥On obtient une grammaire $\mathcal{G}_i$ telle que les variables $X_1, \ldots, X_i$ n'apparaissent pas en tête de partie droite de règle

Mise en forme normale de Greibach - Exemple (1/2)

Grammaire de départ, G_0

$$\begin{array}{lcl} A & \rightarrow & AB + a \\ B & \rightarrow & BC + b \\ C & \rightarrow & CA + c \end{array}$$

G_1 après avoir traité A

$$\begin{array}{lcl} A & \rightarrow & aA' + a \\ A' & \rightarrow & BA' + B \\ B & \rightarrow & BC + b \\ C & \rightarrow & CA + c \end{array}$$

Traitement de B

$$\begin{array}{lcl} A & \rightarrow & aA' + a \\ A' & \rightarrow & BA' + B \\ B & \rightarrow & bB' + b \\ B' & \rightarrow & CB' + C \\ C & \rightarrow & CA + c \end{array}$$

Remplacement de B obtention de G_2

$$\begin{array}{lcl} A & \rightarrow & aA' + a \\ A' & \rightarrow & bB'A' + bA' + bB' + b \\ B & \rightarrow & bB' + b \\ B' & \rightarrow & CB' + C \\ C & \rightarrow & CA + c \end{array}$$

Traitement de C

$$\begin{array}{lcl} A & \rightarrow & aA' + a \\ A' & \rightarrow & bB'A' + bA' + bB' + b \\ B & \rightarrow & bB' + b \\ B' & \rightarrow & CB' + C \\ C & \rightarrow & cC' + c \\ C' & \rightarrow & AC' + A \end{array}$$

Mise en forme normale de Greibach - Exemple (1/2)

$$\begin{array}{lcl}
A & \rightarrow & aA' + a \\
A' & \rightarrow & bB'A' + bA' + bB' + b \\
B & \rightarrow & bB' + b \\
B' & \rightarrow & CB' + C \\
C & \rightarrow & cC' + c \\
C' & \rightarrow & AC' + A
\end{array}$$

Remplacement pour obtenir G_2

$$\begin{array}{lcl}
A & \rightarrow & aA' + a \\
A' & \rightarrow & bB'A' + bA' + bB' + b \\
B & \rightarrow & bB' + b \\
B' & \rightarrow & cC'B' + cB' + cC' + c \\
C & \rightarrow & cC' + c \\
C' & \rightarrow & aA'C' + aC' + aA' + a
\end{array}$$

Exemple de grammaire non $LL(1)$

Exemple de grammaire sur $\Sigma = \{a, b\}$

$$\begin{array}{rcl} S & \to & aSb \mid T \\ T & \to & aT \mid \varepsilon \end{array}$$

Cette grammaire :

- est non ambigüe
- n'est pas récursive à gauche
- est factorisée à gauche

Exemple de grammaire non $LL(1)$

Exemple de grammaire sur $\Sigma = \{a, b\}$

$$
\begin{array}{lcl}
S & \rightarrow & aSb \mid T \\
T & \rightarrow & aT \mid \varepsilon
\end{array}
$$

Cette grammaire :

- est non ambigüe
- n'est pas récursive à gauche
- est factorisée à gauche

Pourtant la table de prédiction

	a	b
S	aSb T	
T	aT	

nous montre qu'elle n'est pas $LL(1)$

Analyse Syntaxique

Analyse $LL(k)$

Analyse $LL(k)$

Parfois les transformations de grammaires que nous connaissons ne suffisent pas à rendre la grammaire $LL(1)$ et il est alors nécessaire de considérer des analyses $LL(k)$ avec $k > 1$

En particulier, pour une grammaire donnée, on ne peut pas décider s'il existe une grammaire équivalente qui soit $LL(1)$

On généralise les définitions utilisées pour l'analyse $LL(1)$ (ou plus exactement celles-ci étaient une spécialisation de la définition générale)

On obtient une hiérarchie stricte des langages $LL(k)$

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire algébrique, pour tout mot $\alpha \in (\Sigma \cup V)^*$, on définit $\mathit{First}_k(\alpha)$ de la manière suivante :

- si $\alpha \in \Sigma^*$ et $|\alpha| \leq k$ alors $\mathit{First}_k(\alpha) = \alpha$
- si $\alpha \in \Sigma^*$ et $|\alpha| > k$ alors $\mathit{First}_k(\alpha) = x_1 \ldots x_k$ avec $\alpha = x_1 \ldots x_k \alpha'$ et $\forall 1 \leq i \leq k, x_i \in \Sigma$
- sinon $\mathit{First}_k(\alpha) = \mathit{First}_k(\mathcal{L}_{\mathcal{G}}(\alpha))$

<u>Concaténation</u>

Par construction $\mathit{First}_k(\alpha\beta) = \mathit{First}_k(\mathit{First}_k(\alpha)\mathit{First}_k(\beta))$

First_k - Calcul

On suppose que les règles de $\mathcal{G}$ sont toutes productives

Pour un symbole $\alpha \in \Sigma \cup V$, on définit $U_n(\alpha)$ pour $n \geq 0$ de la manière suivante :

- si $\alpha \in \Sigma$, $U_n(\alpha) = \{\alpha\}$ pour tout n
- si $\alpha \in V$, on définit U_n inductivement

$$U_n(\alpha) = \bigcup_{\alpha \to \alpha_1 \ldots \alpha_m \in R} \mathit{First}_k(U_{n-1}(\alpha_1) \ldots U_{n-1}(\alpha_m))$$

  en posant $U_0(\alpha) = \emptyset$

Il est ensuite possible de calculer les U_n par point fixe pour tout $\alpha \in V$

$$\mathit{First}_k(\alpha) = \bigcup_{0 \leq n} U_n(\alpha)$$

Reprenons l'exemple du calcul de $First_1$

Exemple de grammaire

$$
\begin{array}{lcl}
S & \rightarrow & TU \mid Xc \\
T & \rightarrow & aUb \mid \varepsilon \\
U & \rightarrow & bX \mid \varepsilon \\
X & \rightarrow & dX \mid e
\end{array}
$$

Déroulons l'algorithme de calcul de *First* pour les non-terminaux

	S	T	U	X
1.1 1.3 2	$\cup First(T) \cup First(X)$ $\varepsilon \in First(T) \Rightarrow \cup First(U)$	$\cup First(a)$ $\cup\{\varepsilon\}$	$\cup First(b)$ $\cup\{\varepsilon\}$	$\cup First(d) \cup First$
It $1 = U_1$	$\emptyset$	$\{a, \varepsilon\}$	$\{b, \varepsilon\}$	$\{d, e\}$
It $2 = U_2$	$\{a, \varepsilon, d, e, b\}$	$\{a, \varepsilon\}$	$\{b, \varepsilon\}$	$\{d, e\}$
It $3 = U_3$	$\{a, \varepsilon, d, e, b\}$	$\{a, \varepsilon\}$	$\{b, \varepsilon\}$	$\{d, e\}$

Exemple de calcul de $First_3$

Exemple de grammaire

$$\begin{array}{lcl} S & \rightarrow & TV \\ T & \rightarrow & ab \mid adeT \\ V & \rightarrow & abc \end{array}$$

Calculons les $First_3$ pour les non-terminaux

$U_n(S) = First_3(U_{n-1}(T)U_{n-1}(V))$

$U_n(T) = First_3(U_{n-1}(a)U_{n-1}(b)) \cup First_3(U_{n-1}(a)U_{n-1}(d)U_{n-1}(e)U_{n-1}(T))$

$U_n(V) = First_3(U_{n-1}(a)U_{n-1}(b)U_{n-1}(c))$

	S	T	V	a	b	c	d	e
U_0	$\emptyset$	$\emptyset$	$\emptyset$	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$
U_1	$\emptyset$	$\{ab, ade\}$	$\{abc\}$	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$
U_2	$\{aba, ade\}$	$\{ab, ade\}$	$\{abc\}$	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$
U_3	$\{aba, ade\}$	$\{ab, ade\}$	$\{abc\}$	$\{a\}$	$\{b\}$	$\{c\}$	$\{d\}$	$\{e\}$

$Follow_k$ - Définition

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire algébrique, soit $T \in V$ alors

$$Follow_k(T) = \{u \in \Sigma^* \mid \exists S \underset{*}{\rightarrow} \alpha T \beta \text{ avec } u \in First_k(\beta)\}$$

i.e. L'idée est la même que pour $Follow_1$: on cherche l'ensemble des k (au plus) symboles terminaux qui peuvent suivre le mot produit par la dérivation de la variable T

<u>Calcul par point fixe</u> On définit les $U_n(T)$ pour tout T accessible :

- $U_0(S) = \{\varepsilon\}$ et $U_0(T) = \emptyset$ pour $T \neq S$
- $U_n(T) = U_{n-1}(T) \cup \bigcup_{X \to \alpha T \beta \in R} First_k(First_k(\beta) U_{n-1}(X))$

Grammaire $LL(k)$

Une grammaire algébrique $\mathcal{G} = (\Sigma, V, S, R)$ est $LL(k)$ si et seulement si pour toute dérivation $S \xrightarrow[*]{} \alpha T \beta$ avec $T \in V$ et pour toute paire de règles $T \rightarrow \gamma$ et $T \rightarrow \delta$, on vérifie

$$First_k(\gamma\beta) \cap First_k(\delta\beta) = \emptyset$$

Une grammaire algébrique $\mathcal{G} = (\Sigma, V, S, R)$ est **fortement** $LL(k)$ si et seulement si pour toute dérivation pour toute paire de règles $T \rightarrow \gamma$ et $T \rightarrow \delta$, on vérifie

$$First_k(\gamma Follow_k(T)) \cap First_k(\delta Follow_k(T)) = \emptyset$$

Table de prédiction et analyse $LL(k)$

Soit $\mathcal{G} = (\Sigma, V, S, R)$ une grammaire fortement $LL(k)$, on définit la **table de prédiction** M_k de la manière suivante

Pour tout $T \in V$ et tout mot $u \in \Sigma^*$ avec $|u| \leq k$

$$M[T, u] = \alpha \text{ avec } T \to \alpha \in R \text{ et } u \in First_k(\alpha Follow_k(T))$$

Les cases vides représentent les cas d'erreur

L'**analyseur** $LL(k)$ est l'automate à pile déterministe qui accepte par pile vide et dont les transitions sont pilotées par la table de prédiction

Proposition

Soit $\mathcal{G}$ une grammaire fortement $LL(k)$ l'analyseur $LL(k)$ de $\mathcal{G}$ reconnait exactement $\mathcal{L}(\mathcal{G})$

Analyse Syntaxique

Exemple de conclusion

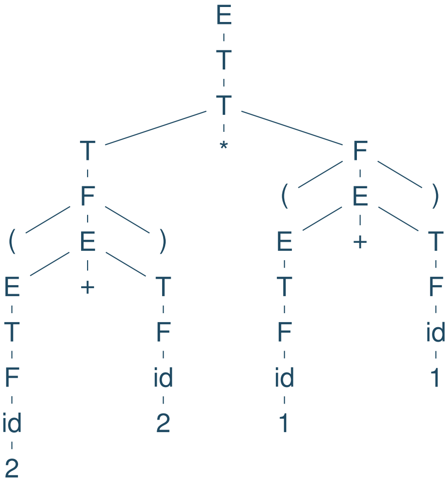
Définition de l'exemple

Reprenons l'exemple des expressions arithmétiques (Dragon Book) auquel on ajoute une règle pour « simuler » des dérivations complètes

Grammaire

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid id \\
id & \rightarrow & 1 \mid 2
\end{array}
$$

Expression $(2 + 2) * (1 + 1)$



Suppression de la récursivité gauche

Grammaire de départ

$$
\begin{array}{lcl}
E & \rightarrow & E + T \mid T \\
T & \rightarrow & T * F \mid F \\
F & \rightarrow & (E) \mid id \\
id & \rightarrow & 1 \mid 2
\end{array}
$$

Grammaire obtenue à la section précédente

$$
\begin{array}{lcl}
E & \rightarrow & TE' \\
E' & \rightarrow & +TE' \mid \varepsilon \\
T & \rightarrow & FT' \\
T' & \rightarrow & *FT' \mid \varepsilon \\
F & \rightarrow & (E) \mid id \\
id & \rightarrow & 1 \mid 2
\end{array}
$$

Calcul de *First*

$$
\begin{array}{lcllcl}
E & \rightarrow & TE' & T' & \rightarrow & *FT' \mid \varepsilon \\
E' & \rightarrow & +TE' \mid \varepsilon & F & \rightarrow & (E) \mid id \\
T & \rightarrow & FT' & id & \rightarrow & 1 \mid 2
\end{array}
$$

E	E'	T	T'	F	id
$First(T)$	$\{+, \varepsilon\}$	$First(F)$	$\{*, \varepsilon\}$	$\{(\} \cup First(id)$	$\{1, 2\}$
	$\{+, \varepsilon\}$		$\{*, \varepsilon\}$	$\{(, 1, 2\}$	$\{1, 2\}$
	$\{+, \varepsilon\}$	$\{(, 1, 2\}$	$\{*, \varepsilon\}$	$\{(, 1, 2\}$	$\{1, 2\}$
$\{(, 1, 2\}$	$\{+, \varepsilon\}$	$\{(, 1, 2\}$	$\{*, \varepsilon\}$	$\{(, 1, 2\}$	$\{1, 2\}$

Calcul de *Follow*

$$
\begin{array}{lllllll}
E & \rightarrow & TE' & \quad & T' & \rightarrow & *FT' \mid \varepsilon \\
E' & \rightarrow & +TE' \mid \varepsilon & \quad & F & \rightarrow & (E) \mid id \\
T & \rightarrow & FT' & \quad & id & \rightarrow & 1 \mid 2
\end{array}
$$

E	E'	T	T'	F	id
$\{\#,)\}$	$Follow(E)$	$First(E')$ $Follow(E)$ $Follow(E')$	$Follow(T)$	$First(T')$ $Follow(T)$ $Follow(T')$	$Follow(F)$
$\{\#,)\}$	$\{\#,)\}$	$\{+, \#,)\}$	$\{+, \#,)\}$	$\{*, +, \#,)\}$	$\{*, +, \#,)\}$

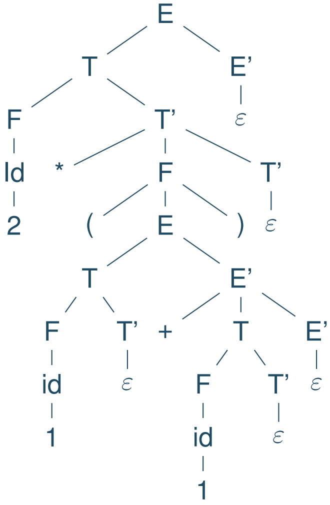
Calcul de la table de prédiction

$$\begin{array}{lllllll} E & \rightarrow & TE' & \quad & T' & \rightarrow & *FT' \mid \varepsilon \\ E' & \rightarrow & +TE' \mid \varepsilon & & F & \rightarrow & (E) \mid id \\ T & \rightarrow & FT' & & id & \rightarrow & 1 \mid 2 \end{array}$$

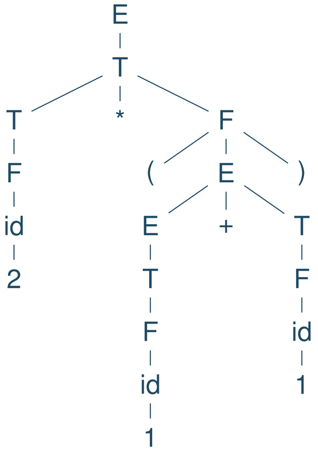
	(	)	1	2	+	*	#
E	TE'		TE'	TE'			
E'					$+TE'$		ε
T	FT'		FT'	FT'			
T'						$*FT'$	ε
F	(E)		id	id			
id			1	2			

Mot à analyser

Exemple : $2 * (1 + 1)$



Dans la grammaire avant transformation



Analyse du mot $2 * (1 + 1)$

	(	)	1	2	$+$	$*$	$\#$
E	TE'		TE'	TE'			
E'					$+TE'$		ε
T	FT'		FT'	FT'			
T'						$*FT'$	ε
F	(E)		id	id			
id			1	2			

Mot à analyser

$2 \quad * \quad (\quad 1 \quad + \quad 1 \quad)$

$\uparrow$

Rappel de la fonction de transition

$$\delta = \{(q_0, z) \xrightarrow{\varepsilon} (q_0, z') \mid z \to z' \in R\} \cup \{(q_0, x) \xrightarrow{x} (q_0, \varepsilon) \mid x \in \Sigma\}$$