Théorie des langages : THL

CM 4

Uli Fahrenberg

EPITA Rennes

S5 2021

Aperçu

Programme du cours

1. Langages rationnels
2. Automates finis
3. Langages algébriques, grammaires hors-contexte, automates à pile
4. Parsage LL, partie 1
5. Parsage LL, partie 2
6. TP 1 : `flex`
7. Parsage LR
8. TP 2, 3 : `flex` & `bison`

La dernière fois : hiérarchie de Chomsky

Une grammaire (syntagmatique) : (N, Σ, P, S) :

- N, Σ : ensembles finis de variables et terminaux
- $S \in N$: le symbole initial
- $P \subseteq (N \cup \Sigma)^+ \times (N \cup \Sigma)^*$: l'ensemble de productions

type	grammaires	productions	langages	automates
4	finis	$N \to \Sigma^*$	finis	finis acycliques
	$\Downarrow$		$\subsetneq$	
3	régulières	$N \to \Sigma^* \cup \Sigma^* N$	réguliers	finis
	$\Downarrow$		$\subsetneq$	
2	hors-contexte	$N \to V^*$	algébriques	à pile
	$\Downarrow$		$\subsetneq$	
1	contextuelles	$\alpha N \beta \to \alpha V^+ \beta$ $S_0 \to S \mid \varepsilon$	contextuels	linéairement bornés
	$\Downarrow$		$\subsetneq$	
0	syntagmatiques	$V^+ \to V^*$	récursivement énumerables	de Turing

Aujourd'hui : parsage LL

Une grammaire hors contexte : (N, Σ, P, S) :

- N, Σ : ensembles finis de variables et terminaux
- $S \in N$: le symbole initial
- $P \subseteq (N \times (N \cup \Sigma)^*$: l'ensemble de productions

But : construire des algorithmes de parsage basés sur grammaires hors-contexte

- grammaire hc $G \rightsquigarrow$ algorithme A
- A : mot $w \rightsquigarrow$ reject / accept
- faut que A soit efficace
- dans le poly : section 6.2, chapitre 7, section 8.1

Aujourd'hui : parsage LL

Une grammaire hors contexte : (N, Σ, P, S) :

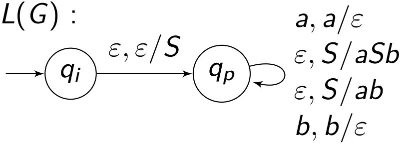
- N, Σ : ensembles finis de variables et terminaux
- $S \in N$: le symbole initial
- $P \subseteq (N \times (N \cup \Sigma)^*$: l'ensemble de productions

But : construire des algorithmes de parsage basés sur grammaires hors-contexte

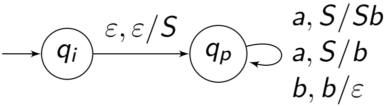
- YACC : grammaire hc $G \rightsquigarrow$ algorithme A
- A : mot $w \rightsquigarrow$ reject / accept + arbre de parsage
- faut que A soit efficace
- dans le poly : section 6.2, chapitre 7, section 8.1

Exemple

- $G : S \to aSb \mid ab \qquad L(G) = \{a^n b^n \mid n \geq 1\}$
- automate à pile standard pour $L(G)$:



- plein de transitions spontanées, plein de non-déterminisme
- automate à pile type Greibach :



- déjà mieux, mais toujours plein de non-déterminisme
- (on s'en fout de la première transition spontanée)

But : algorithmes de parsage déterministes en temps linéaire

Aussi la dernière fois : langages de Dyck

- langages des mot bien parenthésés
- grammaire de Dyck d'ordre n :

$$G_n : S \to a_1 S b_1 \mid a_2 S b_2 \mid \cdots \mid a_n S b_n \mid SS \mid \varepsilon$$

- (avec $\Sigma = \{a_i, b_i \mid i = 1, \dots, n\}$:
- $a_i - b_i$ parenthèses correspondantes)
- $L(G_1) = \{w \in \{a, b\}^* \mid |w|_a = |w|_b \wedge \forall u \in \mathrm{Pref}(w) : |u|_a \geq |u|_b\}$

Dérivations

Arbres de dérivation

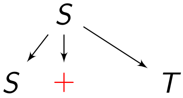
Quelques expressions arithmétiques :

$$
\begin{aligned}
S &\to S + T \mid T \\
T &\to T * F \mid F \\
F &\to (S) \mid a
\end{aligned}
$$

Arbres de dérivation

Quelques expressions arithmétiques :

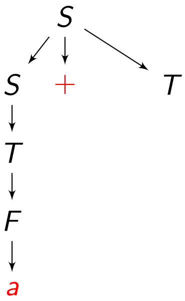
$$S \to S + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (S) \mid a$$



Arbres de dérivation

Quelques expressions arithmétiques :

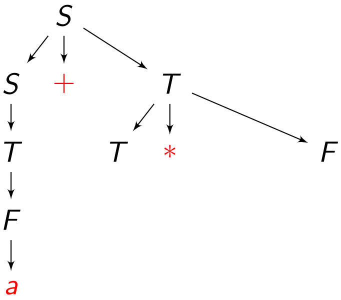
$$
\begin{aligned}
S &\to S + T \mid T\\
T &\to T * F \mid F\\
F &\to (S) \mid a
\end{aligned}
$$



Arbres de dérivation

Quelques expressions arithmétiques :

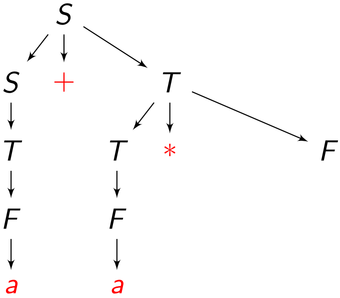
$$S \to S + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (S) \mid a$$



Arbres de dérivation

Quelques expressions arithmétiques :

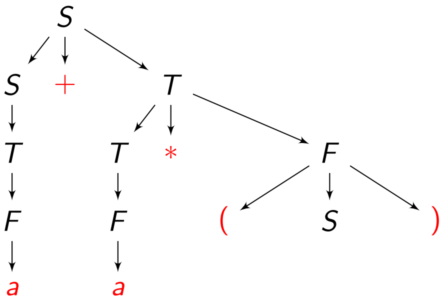
$$S \to S + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (S) \mid a$$



Arbres de dérivation

Quelques expressions arithmétiques :

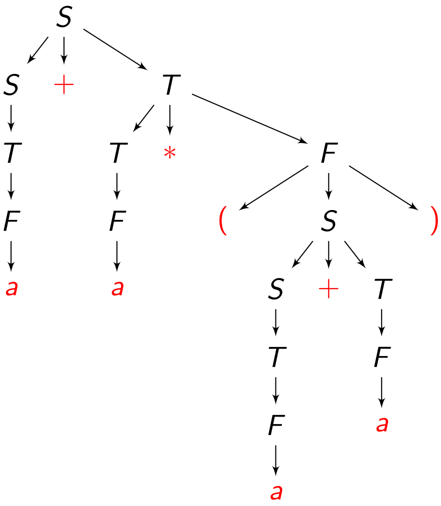
$$S \to S + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (S) \mid a$$



Arbres de dérivation

Quelques expressions arithmétiques :

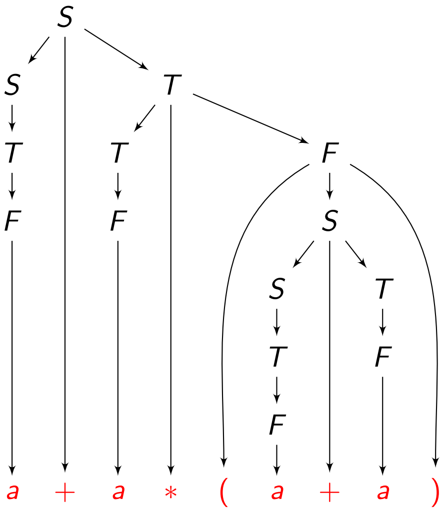
$$S \rightarrow S + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (S) \mid a$$



Arbres de dérivation

Quelques expressions arithmétiques :

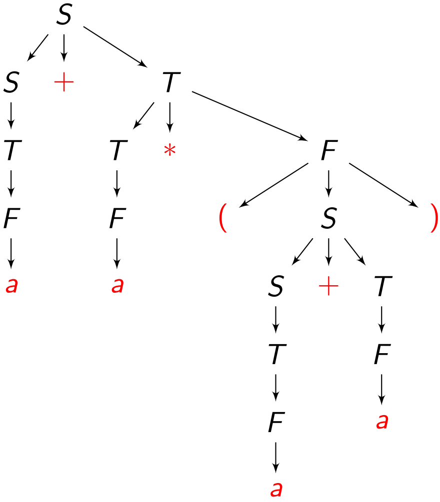
$$S \to S + T \mid T$$
$$T \to T * F \mid F$$
$$F \to (S) \mid a$$



Arbres de dérivation

Quelques expressions arithmétiques :

$$
\begin{aligned}
S &\to S + T \mid T \\
T &\to T * F \mid F \\
F &\to (S) \mid a
\end{aligned}
$$



- plusieurs dérivations, même arbre :
- $S \Rightarrow S + T \Rightarrow T + T \Rightarrow F + T \Rightarrow a + T \Rightarrow \dots$
- $S \Rightarrow S + T \Rightarrow S + T * F \Rightarrow S + F * F \Rightarrow \dots$
- etc.
- on s'intéresse aux arbres, pas aux dérivations

Dérivations gauche

Soit G une grammaire hors-contexte.

Définition (6.1)

Une dérivation $S \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow w$ dans G est dite gauche si à chaque pas $\alpha_i \Rightarrow \alpha_{i+1}$ c'est la variable la plus à gauche dans α_i qui est réécrit.

- par analogie, aussi « dérivation droite »

Théorème (6.3)

Pour chaque $w \in L(G)$ il existe une dérivation gauche $S \Rightarrow^* w$.

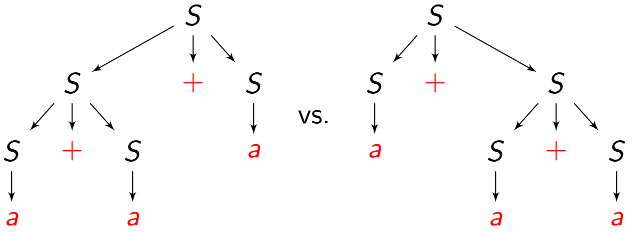
Ambiguité

Exemple : $G : S \to S + S \mid a$

deux dérivations gauche différents :

- $S \Rightarrow S + S \Rightarrow S + S + S \Rightarrow a + S + S \Rightarrow a + a + S \Rightarrow a + a + a$
- $S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S + S \Rightarrow a + a + S \Rightarrow a + a + a$

correspondant à deux arbres différents :



- $(a + a) + a$ vs. $a + (a + a)$
- heureusement l'addition est associative !

Associativité de l'addition (ou pas)

Addition des `Int32` :

$$x = 2^{31} - 1 \qquad y = 1 \qquad z = -1$$

$$x + (y + z) = x + 0 = 2^{31} - 1$$

$$(x + y) + z = 0 + z = -1$$

- overflow !

Ambiguité

Définition (6.5)

Une grammaire hors-contexte est ambiguë s'il existe $w \in L(G)$ admettant deux dérivations gauches différents.

- équivalent : « . . . admettant deux arbres de dérivation différents »
- deux arbres différents $\Rightarrow$ deux sémantiques différents

$\Rightarrow$ pour le parsage, faut des grammaires non-ambiguës

Théorème (sans démonstration ici)

Il existe des langages algébriques qui ne peuvent être engendrés que par des grammaires ambiguës.

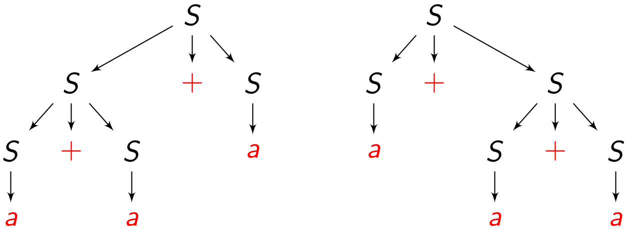
- par exemple $L = \{a^m b^n c^p \mid m = n \text{ ou } n = p\}$
- un langage intrinsèquement ambigu
- on ne peut pas traiter des langages intrinsèquement ambigus

Fin à l'ambiguité

Dans la pratique il existe toujours des grammaires hc non-ambiguës.

Exemple :

- $S \to S + S \mid a \quad \rightsquigarrow \quad S \to S + a \mid a$
- engendre le même langage, avec associativité à gauche



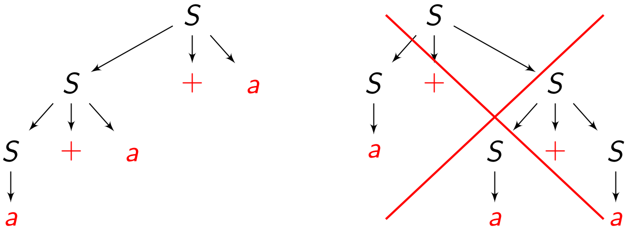
- $S \to S + T \mid T$; $T \to T * F \mid F$; $F \to (S) \mid a$: même chose

Fin à l'ambiguité

Dans la pratique il existe toujours des grammaires hc non-ambiguës.

Exemple :

- $S \to S + S \mid a \quad \rightsquigarrow \quad S \to S + a \mid a$
- engendre le même langage, avec associativité à gauche



- $S \to S + T \mid T\,;\ T \to T * F \mid F\,;\ F \to (S) \mid a$: même chose

Parsage LL(1)

Parsage

Problème de parsage

Pour une grammaire hc G, construire un algorithme qui :

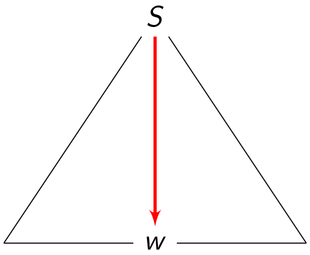
- pour un mot w, decide si $w \in L(G)$
- et dans le cas $w \in L(G)$, retourne l'arbre de dérivation

- arbre de dérivation de w $\hat{=}$ sémantique de w

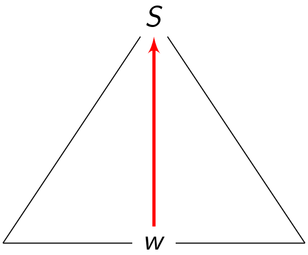
Nos algorithmes de parsage devrait

- pouvoir traiter des grammaires non-ambiguës
- avoir une complexité linéaire en taille d'entrée
- lire w de gauche à droite sans retour arrière

Approches



descendante ↑ maintenant



ascendante ↑ plus tard

Exemple

$$\begin{aligned} S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1) \\ &\mid \texttt{echo} && (2) \end{aligned} \qquad\qquad \begin{aligned} E &\to \texttt{true} && (3) \\ &\mid \texttt{false} && (4) \end{aligned}$$

Mot d'entrée :

`if true then if false then echo fi fi`

Arbre construit :

S

Exemple

$$
\begin{aligned}
S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1) \qquad & E &\to \texttt{true} && (3)\\
&\mid \texttt{echo} && (2) \qquad & &\mid \texttt{false} && (4)
\end{aligned}
$$

Mot d'entrée :

`if true then if false then echo fi fi`

Arbre construit :

S

Exemple

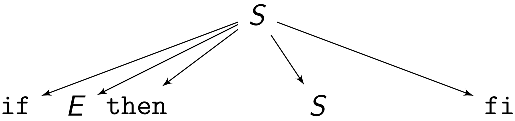
$$\begin{aligned} S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1)\\ &\mid \texttt{echo} && (2)\end{aligned}$$

$$\begin{aligned} E &\to \texttt{true} && (3)\\ &\mid \texttt{false} && (4)\end{aligned}$$

Mot d'entrée :

`if true then if false then echo fi fi`
(1)

Arbre construit :



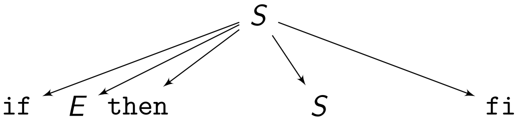
Exemple

$$\begin{aligned} S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1) \qquad\qquad & E &\to \texttt{true} && (3)\\ &\mid \texttt{echo} && (2) & &\mid \texttt{false} && (4) \end{aligned}$$

Mot d'entrée :

$$\texttt{if}\ \underset{(3)}{\texttt{true}}\ \texttt{then if false then echo fi fi}$$

Arbre construit :



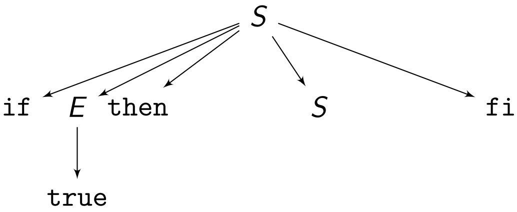
Exemple

$$
\begin{aligned}
S &\to \texttt{if } E \texttt{ then } S \texttt{ fi} && (1) \qquad\qquad & E &\to \texttt{true} && (3)\\
&\mid \texttt{echo} && (2) & &\mid \texttt{false} && (4)
\end{aligned}
$$

Mot d'entrée :

$\underset{}{\texttt{if}}\ \underset{(3)}{\texttt{true}}\ \texttt{then if false then echo fi fi}$

Arbre construit :



S

if E then S fi

true

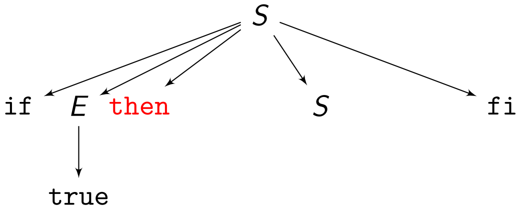
Exemple

$$
\begin{aligned}
S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1) & \qquad E &\to \texttt{true} && (3)\\
&\mid \texttt{echo} && (2) & &\mid \texttt{false} && (4)
\end{aligned}
$$

Mot d'entrée :

Arbre construit :

`if true then if false then echo fi fi`



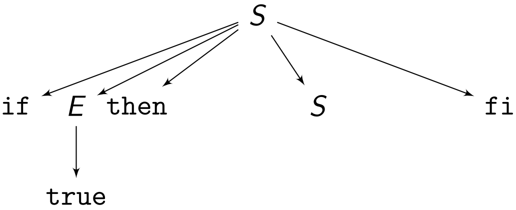
Exemple

$$\begin{aligned} S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1) \qquad & E &\to \texttt{true} && (3)\\ &\mid \texttt{echo} && (2) \qquad & &\mid \texttt{false} && (4) \end{aligned}$$

Mot d'entrée :

Arbre construit :

$\texttt{if true then } \underset{(1)}{\texttt{if}} \texttt{ false then echo fi fi}$



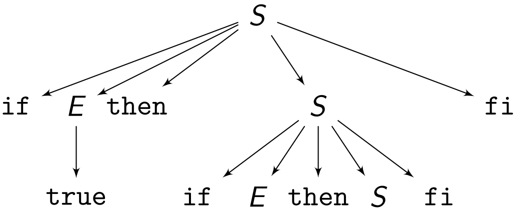
Exemple

$$
\begin{aligned}
S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1) \qquad\qquad & E &\to \texttt{true} && (3)\\
&\mid \texttt{echo} && (2) & &\mid \texttt{false} && (4)
\end{aligned}
$$

Mot d'entrée :

`if true then` $\underset{(1)}{\texttt{if}}$ `false then echo fi fi`

Arbre construit :



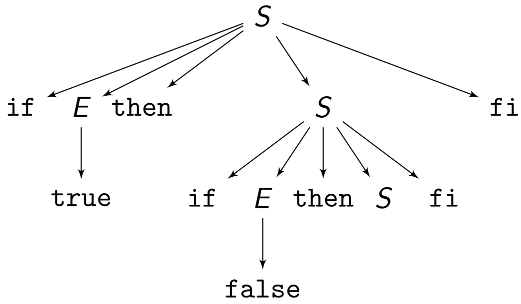
Exemple

$$\begin{aligned} S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1) \qquad & E &\to \texttt{true} && (3) \\ &\mid \texttt{echo} && (2) \qquad & &\mid \texttt{false} && (4) \end{aligned}$$

Mot d'entrée :

$\texttt{if true then if } \underset{(4)}{\texttt{false}} \texttt{ then echo fi fi}$

Arbre construit :



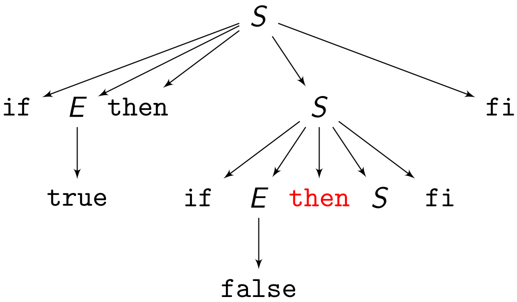
Exemple

$$\begin{array}{llr@{\qquad\qquad}llr}
S \to & \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} & (1) & E \to & \texttt{true} & (3) \\
\mid & \texttt{echo} & (2) & \mid & \texttt{false} & (4)
\end{array}$$

Mot d'entrée :

`if true then if false then echo fi fi`

Arbre construit :



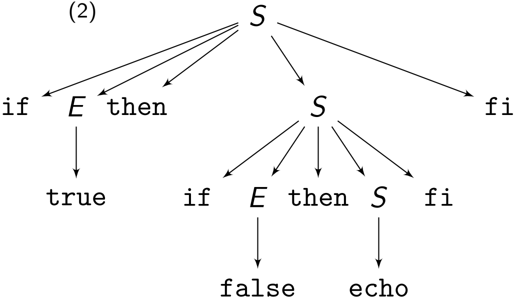
Exemple

$$\begin{aligned} S &\to \texttt{if } E \texttt{ then } S \texttt{ fi} && (1) \\ &\mid \texttt{echo} && (2) \end{aligned} \qquad \begin{aligned} E &\to \texttt{true} && (3) \\ &\mid \texttt{false} && (4) \end{aligned}$$

Mot d'entrée :

Arbre construit :

$\texttt{if true then if false then } \underset{(2)}{\texttt{echo}} \texttt{ fi fi}$



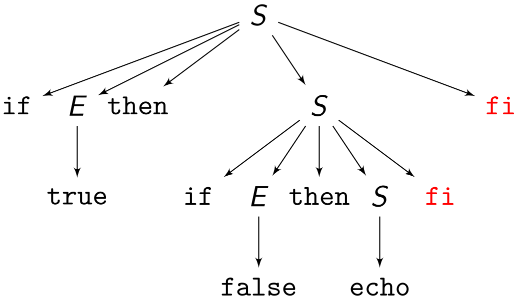
Exemple

$$\begin{aligned} S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1) \\ &\mid \texttt{echo} && (2) \end{aligned} \qquad \begin{aligned} E &\to \texttt{true} && (3) \\ &\mid \texttt{false} && (4) \end{aligned}$$

Mot d'entrée :

Arbre construit :

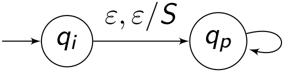
`if true then if false then echo fi fi`



Exemple, version Greibach

$S \to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} \mid \texttt{echo}$

$E \to \texttt{true} \mid \texttt{false}$



- une grammaire Greibach déterministe

état	pile	reste du mot
q_i	ε	`if true then if false then echo fi fi`
q_p	S	`if true then if false then echo fi fi`
q_p	E `then` S `fi`	`true then if false then echo fi fi`
q_p	`then` S `fi`	`then if false then echo fi fi`
q_p	S `fi`	`if false then echo fi fi`
q_p	E `then` S `fi fi`	`false then echo fi fi`
q_p	`then` S `fi fi`	`then echo fi fi`
q_p	...	...

Parsage LL(1)

- approche descendante
- lire le mot *w* de gauche à droite / Left-to-right
  - sans passer à l'arrière
- construire une dérivation gauche / Leftmost
- en accordant, à chaque pas, le premier symbole de *w* avec le côté droit d'une production

Exemple, encore

$$S \to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} \quad (1) \qquad E \to \texttt{true} \quad (3)$$

$$\mid \texttt{echo} \quad (2) \qquad \mid \texttt{false} \quad (4)$$

Une table de parsage :

	`if`	`then`	`fi`	`echo`	`true`	`false`
S	1			2		
E					3	4

- case vide : erreur de parsage

End of file

C'est souhaitable de pouvoir expliciter la fin d'entrée.

- on utilise « \$ » comme symbole EOF ici
- `if true then if false then echo fi fi$`

Pour adapter la grammaire :

- rajout d'une nouvelle variable Z
- avec production $Z \to S\$$

End of file

C'est souhaitable de pouvoir expliciter la fin d'entrée.

- on utilise « \$ » comme symbole EOF ici
- `if true then if false then echo fi fi$`

Pour adapter la grammaire :

- rajout d'une nouvelle variable Z
- avec production $Z \to S\$$

Exemple :

$$S \to \texttt{if } E \texttt{ then } S \texttt{ fi} \quad (1) \qquad E \to \texttt{true} \quad (3)$$

$$\mid \texttt{echo} \quad (2) \qquad \mid \texttt{false} \quad (4)$$

End of file

C'est souhaitable de pouvoir expliciter la fin d'entrée.

- on utilise « \$ » comme symbole EOF ici
- `if true then if false then echo fi fi$`

Pour adapter la grammaire :

- rajout d'une nouvelle variable Z
- avec production $Z \to S\$$

Exemple :

$$\begin{aligned} Z &\to S\$ && (0) \\ S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1) \\ &\mid \texttt{echo} && (2) \end{aligned} \qquad \begin{aligned} E &\to \texttt{true} && (3) \\ &\mid \texttt{false} && (4) \end{aligned}$$

FIRST

$$
\begin{array}{rll}
Z \to S\$ & & (0) \\
S \to \texttt{if } E \texttt{ then } S \texttt{ fi} & & (1) \\
\mid \texttt{echo} & & (2)
\end{array}
\qquad
\begin{array}{rll}
E \to \texttt{true} & & (3) \\
\mid \texttt{false} & & (4)
\end{array}
$$

Pour construire la table de parsage, on a besoin de savoir quel peut être les terminaux à gauche d'une dérivation depuis une variable.

Définition (8.2)

Soit $A \in N$, alors $\text{FIRST}(A) \subseteq \Sigma$ est defini par
$\text{FIRST}(A) = \{a \in \Sigma \mid \exists w \in V^* : A \Rightarrow^* aw\}$.

```
def FIRST(A):
    res = {}
    foreach (A to aw):
        res += {a}
    foreach (A to Bw):
        res += FIRST(B)
    return res
```

A	FIRST(A)
Z	
S	
E	

FIRST

$$\begin{array}{rlr@{\qquad\qquad}rlr}
Z & \to S\$ & (0) & & & \\
S & \to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} & (1) & E & \to \texttt{true} & (3) \\
 & \mid \texttt{echo} & (2) & & \mid \texttt{false} & (4)
\end{array}$$

Pour construire la table de parsage, on a besoin de savoir quel peut être les terminaux à gauche d'une dérivation depuis une variable.

Définition (8.2)

Soit $A \in N$, alors $\mathrm{FIRST}(A) \subseteq \Sigma$ est defini par
$\mathrm{FIRST}(A) = \{a \in \Sigma \mid \exists w \in V^* : A \Rightarrow^* aw\}$.

```
def FIRST(A):
    res = {}
    foreach (A to aw):
        res += {a}
    foreach (A to Bw):
        res += FIRST(B)
    return res
```

A	$\mathrm{FIRST}(A)$
Z	
S	`if`, `echo`
E	`true`, `false`

FIRST

$$
\begin{aligned}
Z &\to S\$ && (0) \\
S &\to \texttt{if}\ E\ \texttt{then}\ S\ \texttt{fi} && (1) \\
&\mid \texttt{echo} && (2)
\end{aligned}
\qquad
\begin{aligned}
E &\to \texttt{true} && (3) \\
&\mid \texttt{false} && (4)
\end{aligned}
$$

Pour construire la table de parsage, on a besoin de savoir quel peut être les terminaux à gauche d'une dérivation depuis une variable.

Définition (8.2)

Soit $A \in N$, alors FIRST(A) $\subseteq \Sigma$ est defini par
$\mathrm{FIRST}(A) = \{a \in \Sigma \mid \exists w \in V^* : A \Rightarrow^* aw\}$.

```
def FIRST(A):
    res = {}
    foreach (A to aw):
        res += {a}
    foreach (A to Bw):
        res += FIRST(B)
    return res
```

A	FIRST(A)
Z	`if`, `echo`
S	`if`, `echo`
E	`true`, `false`

FIRST problems

```
def FIRST(A):
    res = {}
    foreach (A to aw):
        res += {a}
    foreach (A to Bw):
        res += FIRST(B)
    return res
```

Un algorithme de point fixe

FIRST problems

```
def FIRST(A):
    res = {}
    foreach (A to aw):
        res += {a}
    foreach (A to Bw):
        res += FIRST(B)
    return res
```

Un algorithme de point fixe

- mais si $A \to Aw$?
    - récursion à gauche : on ne l'aime pas, faut éviter

FIRST problems

```
def FIRST(A):
    res = {}
    foreach (A to aw):
        res += {a}
    foreach (A to Bw):
        res += FIRST(B)
    return res
```

Un algorithme de point fixe

- mais si $A \to Aw$?
  - récursion à gauche : on ne l'aime pas, faut éviter
- ou si $A \to Bw$ et $B \Rightarrow \varepsilon$?
  - traiter avec NULL et FOLLOW, plus tard
  - pour le moment, ignorer

Algorithme LL(1)

1. entrée : une grammaire hc G
2. construire la table FIRST
3. utiliser FIRST pour construire la TABLE de parsage :

$$\mathrm{TABLE}(A, a) = \{n \mid \exists \alpha \in V, w \in V^* : A \xrightarrow{(n)} \alpha w, a \in \mathrm{FIRST}(\alpha)\}$$

- pour simplicité, $\mathrm{FIRST}(a) = \{a\}$ pour tout $a \in \Sigma$

Définition

G est LL(1) si chaque $\mathrm{TABLE}(A, a)$ contient au maximum une production.

Exemple

$$\mathrm{TABLE}(A, a) = \{n \mid \exists \alpha \in V, w \in V^* : A \xrightarrow{(n)} \alpha w, a \in \mathrm{FIRST}(\alpha)\}$$

$$\begin{aligned}
Z &\to S\$ && (0)\\
S &\to \texttt{if } E \texttt{ then } S \texttt{ fi} && (1)\\
&\mid \texttt{echo} && (2)\\
E &\to \texttt{true} && (3)\\
&\mid \texttt{false} && (4)
\end{aligned}$$

A	FIRST(A)
Z	`if`, `echo`
S	`if`, `echo`
E	`true`, `false`

	`if`	`then`	`fi`	`echo`	`true`	`false`	$
Z	0			0			
S	1			2			
E					3	4	

LL(1)isation

Factorisation

$$
\begin{aligned}
Z &\to S\$ && (0) & S &\to \texttt{echo} && (3)\\
S &\to \texttt{if}\ E\ \texttt{then}\ S && (1) & E &\to \texttt{true} && (4)\\
&\mid \texttt{if}\ E\ \texttt{then}\ S\ \texttt{else}\ S && (2) & &\mid \texttt{false} && (5)
\end{aligned}
$$

	`if`	`then`	`else`	`echo`	`true`	`false`	`$`
Z	0			0			
S	1, 2			3			
E					4	5	

- « conflit FIRST/FIRST »
- notre grammaire n'est pas LL(1)
- solution : factorisation gauche

Factorisation gauche

Théorème (8.6)

Pour chaque grammaire hc G il existe une autre G' avec $L(G) = L(G')$ et telle que pour chaque paire $A \to X_1 \dots X_k \mid Y_1 \dots Y_\ell$ de productions, $X_1 \neq Y_1$.

Exemple :

$$\begin{aligned} S &\to \texttt{if}\ E\ \texttt{then}\ S \\ &\quad \mid \texttt{if}\ E\ \texttt{then}\ S\ \texttt{else}\ S \end{aligned}$$

devient

$$\begin{aligned} S &\to \texttt{if}\ E\ \texttt{then}\ S\,X \\ X &\to \texttt{else}\ S \mid \varepsilon \end{aligned}$$

- attention à la production $X \to \varepsilon$

Fin à la récursion gauche

Sheila Greibach to the rescue !

Théorème (re THL 3)

Pour chaque grammaire hors-contexte G il existe une autre G' telle que $L(G') = L(G)$ et toutes les productions sont sous la forme $S \to \varepsilon$ ou $A \to a\alpha$ avec $a \in \Sigma$ et $\alpha \in (N \setminus \{S\})^*$.

- donc $S \to \varepsilon$ ou $A \to aA_1 \dots A_n$
- convertir récursion gauche en récursion droite

Exemple :

$$\begin{aligned} X &\to Xu \\ &\mid v \end{aligned} \qquad \text{devient} \qquad \begin{aligned} X &\to vY \\ Y &\to uY \\ &\mid \varepsilon \end{aligned}$$

LL(1)isation

Pour convertir G en LL(1) :

- éliminer récursion à gauche pour pouvoir calculer FIRST
- factorisation gauche pour éviter des conflits FIRST/FIRST

- les deux constructions introduit des productions type $A \to \varepsilon$
- alors comment modifier notre algorithme LL(1) pour les traiter ?

FIRST avec ε

$$\text{FIRST}(A) = \{a \in \Sigma \mid \exists w \in V^* : A \Rightarrow^* aw\}$$

```
def FIRST(A):
    res = {}
    foreach (A to aw):
        res += {a}
    foreach (A to Bw):
        res += FIRST(B)
    return res
```

- mais si $A \to Aw$?
    - récursion à gauche : on sais l'éviter
- ou si $A \to Bw$ et $B \Rightarrow \varepsilon$?
    - traiter avec NULL et FOLLOW :

NULL

Définition (8.3)

NULL $\subseteq N$ est défini par NULL $= \{A \in N \mid A \Rightarrow^* \varepsilon\}$.

```
def NULL():
    res = {A | A to epsilon};
    while true:
        new = res
            foreach A to A1...An:
                if all(Ai in new):
                    new += {A}
        if new == res: break
        res = new
    return res
```

- encore un algorithme de point fixe

Exemple

$$\begin{aligned} Z &\to XYZ \mid c \\ X &\to Y \mid a \\ Y &\to b \mid \varepsilon \end{aligned}$$

NULL =

Exemple

$$\begin{aligned} Z &\to XYZ \mid c \\ X &\to Y \mid a \\ Y &\to b \mid \varepsilon \end{aligned}$$

$\mathrm{NULL} = \{X, Y\}$

FIRST avec ε, bis

$$\mathrm{FIRST}(A) = \{a \in \Sigma \mid \exists w \in V^* : A \Rightarrow^* aw\}$$

```
def FIRST(X):
    if X == a: return {a}
    if X == epsilon: return {}
    res = {}
    foreach (X to A1 .. An Y w):
        if all(NULL(Ai)):
            res += FIRST(Y)
    return res
```

Exemple, bis

$$Z \to XYZ \mid c$$
$$X \to Y \mid a \qquad \mathrm{NULL} = \{X, Y\}$$
$$Y \to b \mid \varepsilon$$

A	FIRST(A)
X	a
Y	b
Z	c

Exemple, bis

$$Z \to XYZ \mid c$$
$$X \to Y \mid a \qquad \text{NULL} = \{X, Y\}$$
$$Y \to b \mid \varepsilon$$

A	FIRST(A)
X	a, b
Y	b
Z	c

Exemple, bis

$$\begin{aligned} Z &\to XYZ \mid c \\ X &\to Y \mid a \\ Y &\to b \mid \varepsilon \end{aligned}$$

$$\mathrm{NULL} = \{X, Y\}$$

A	$\mathrm{FIRST}(A)$
X	a, b
Y	b
Z	c, a

Exemple, bis

$$\begin{aligned} Z &\to XYZ \mid c \\ X &\to Y \mid a \\ Y &\to b \mid \varepsilon \end{aligned} \qquad \mathrm{NULL} = \{X, Y\}$$

A	FIRST(A)
X	a, b
Y	b
Z	c, a, b

FOLLOW

Le dernier morceau : calculer des terminaux qui peuvent suivre une variable dans une dérivation :

Définition (8.4, corrigé)

Soit $A \in N$, alors $\text{FOLLOW}(A) \subseteq \Sigma$ est défini par
$\text{FOLLOW}(A) = \{a \in \Sigma \mid \exists B \in N, \alpha, \beta \in V^* : B \Rightarrow^* \alpha A a \beta\}$.

Algorithme :

1. pour chaque $A \in N$: FOLLOW(A) = $\emptyset$
2. répéter jusqu'au point fixe :
    1. pour chaque $B \to \alpha A \beta \gamma$ avec $\beta \in \text{NULL}^*$:
        1. si $\gamma \notin \text{NULL}^*$: FOLLOW(A) += FIRST(γ)
        2. si $\gamma \in \text{NULL}^*$: FOLLOW(A) += FOLLOW(B)

Exemple, bis

$$\begin{aligned} Z &\to XYZ \mid c \\ X &\to Y \mid a \\ Y &\to b \mid \varepsilon \end{aligned}$$

$$\text{NULL} = \{X, Y\}$$

A	FIRST(A)
X	a, b
Y	b
Z	c, a, b

A	FOLLOW(A)
X	
Y	
Z	

Exemple, bis

$$\begin{aligned} Z &\to XYZ \mid c \\ X &\to Y \mid a \\ Y &\to b \mid \varepsilon \end{aligned}$$

$$\text{NULL} = \{X, Y\}$$

A	FIRST(A)
X	a, b
Y	b
Z	c, a, b

A	FOLLOW(A)
X	a, b, c
Y	a, b, c
Z	

Algorithme LL(1) complet

1. entrée : une grammaire hc G
2. calculer NULL
3. construire la table FIRST
4. construire la table FOLLOW
5. construire la TABLE de parsage :
    1. pour chaque production $X \to w \quad (n)$:
        1. pour chaque $a \in \text{FIRST}(w)$: $\text{TABLE}(X, a)$ += $\{n\}$
        2. si $w \in \text{NULL}$ ou $w = \varepsilon$:
           – pour chaque $a \in \text{FOLLOW}(X)$: $\text{TABLE}(X, a)$ += $\{n\}$

Exemple, bis

$$
\begin{array}{rll}
Z \to XYZ & & (1) \\
\mid c & & (2) \\
X \to a & & (3) \\
\mid Y & & (4) \\
Y \to b & & (5) \\
\mid \varepsilon & & (6)
\end{array}
$$

$\text{NULL} = \{X, Y\}$

A	FIRST(A)
X	a, b
Y	b
Z	c, a, b

A	FOLLOW(A)
X	a, b, c
Y	a, b, c
Z	

	a	b	c
X			
Y			
Z			

Exemple, bis

$$\begin{aligned} Z &\to XYZ && (1)\\ &\mid c && (2)\\ X &\to a && (3)\\ &\mid Y && (4)\\ Y &\to b && (5)\\ &\mid \varepsilon && (6) \end{aligned}$$

$\text{NULL} = \{X, Y\}$

A	FIRST(A)
X	a, b
Y	b
Z	c, a, b

A	FOLLOW(A)
X	a, b, c
Y	a, b, c
Z	

	a	b	c
X	3		
Y			
Z			

Exemple, bis

$$\begin{array}{rll} Z \to XYZ & & (1) \\ & \mid c & (2) \\ X \to a & & (3) \\ & \mid Y & (4) \\ Y \to b & & (5) \\ & \mid \varepsilon & (6) \end{array}$$

$$\text{NULL} = \{X, Y\}$$

A	FIRST(A)
X	a, b
Y	b
Z	c, a, b

A	FOLLOW(A)
X	a, b, c
Y	a, b, c
Z	

	a	b	c
X	3, 4	4	4
Y			
Z			

Exemple, bis

$$
\begin{aligned}
Z &\to XYZ && (1)\\
&\mid c && (2)\\
X &\to a && (3)\\
&\mid Y && (4)\\
Y &\to b && (5)\\
&\mid \varepsilon && (6)
\end{aligned}
$$

$$\text{NULL} = \{X, Y\}$$

A	FIRST(A)
X	a, b
Y	b
Z	c, a, b

A	FOLLOW(A)
X	a, b, c
Y	a, b, c
Z	

	a	b	c
X	3, 4	4	4
Y		5	
Z			

Exemple, bis

$$
\begin{array}{rll}
Z \to XYZ & & (1) \\
\mid c & & (2) \\
X \to a & & (3) \\
\mid Y & & (4) \\
Y \to b & & (5) \\
\mid \varepsilon & & (6)
\end{array}
$$

$$\text{NULL} = \{X, Y\}$$

A	FIRST(A)
X	a, b
Y	b
Z	c, a, b

A	FOLLOW(A)
X	a, b, c
Y	a, b, c
Z	

	a	b	c
X	3, 4	4	4
Y	6	5, 6	6
Z			

Exemple, bis

$$\begin{aligned} Z &\to XYZ && (1)\\ &\mid c && (2)\\ X &\to a && (3)\\ &\mid Y && (4)\\ Y &\to b && (5)\\ &\mid \varepsilon && (6) \end{aligned}$$

$$\text{NULL} = \{X, Y\}$$

A	FIRST(A)
X	a, b
Y	b
Z	c, a, b

A	FOLLOW(A)
X	a, b, c
Y	a, b, c
Z	

	a	b	c
X	3, 4	4	4
Y	6	5, 6	6
Z	1	1	1, 2


That's all Folks!