

Théorie des langages : THL

CM 4

Uli Fahrenberg

EPITA Rennes

S5 2024

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 TP 2 : parsage LL
- ⑧ Parsage LR
- ⑨ TP 3, 4 : 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 ↓	$N \rightarrow \Sigma^*$	finis ⋈	finis acycliques
3	régulières ↓	$N \rightarrow \Sigma^* \cup \Sigma^* N$	réguliers ⋈	finis
2	hors-contexte ↓	$N \rightarrow V^*$	algébriques ⋈	à pile
1	contextuelles ↓	$\alpha N \beta \rightarrow \alpha V^+ \beta$ $S_0 \rightarrow S \mid \varepsilon$	contextuels ⋈	linéairement bornés
0	syntagmatiques	$V^+ \rightarrow V^*$	récursivement énumérables	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 : parsing 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**

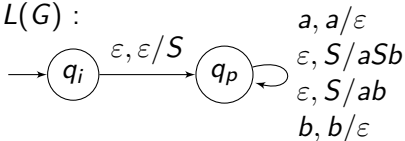
But : construire des algorithmes de **parsage** basés sur grammaires hors-contexte

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

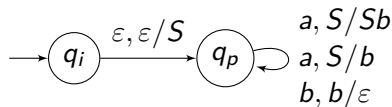
Exemple

- $G : S \rightarrow aSb \mid ab \quad L(G) = \{a^n b^n \mid n \geq 1\}$

- 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**

Dérivations

Arbres de dérivation

Quelques expressions arithmétiques :

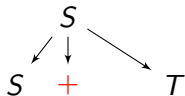
$$S \rightarrow S + T \mid T$$

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

$$F \rightarrow (S) \mid a$$

Arbres de dérivation

Quelques expressions arithmétiques :



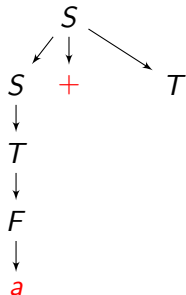
$$S \rightarrow S + T \mid T$$

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

$$F \rightarrow (S) \mid a$$

Arbres de dérivation

Quelques expressions arithmétiques :



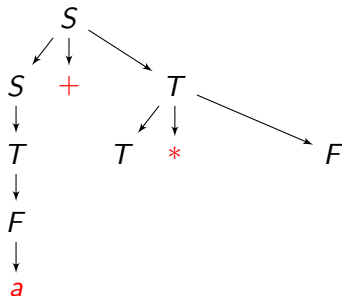
$$S \rightarrow S + T \mid T$$

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

$$F \rightarrow (S) \mid a$$

Arbres de dérivation

Quelques expressions arithmétiques :



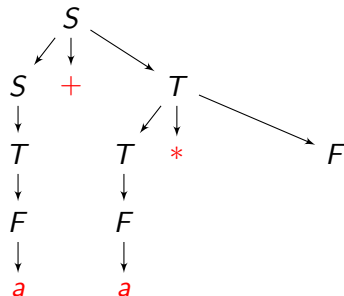
$$S \rightarrow S + T \mid T$$

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

$$F \rightarrow (S) \mid a$$

Arbres de dérivation

Quelques expressions arithmétiques :



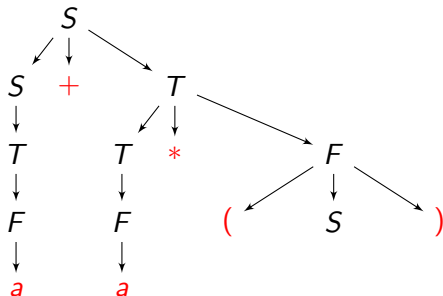
$$S \rightarrow S + T \mid T$$

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

$$F \rightarrow (S) \mid a$$

Arbres de dérivation

Quelques expressions arithmétiques :



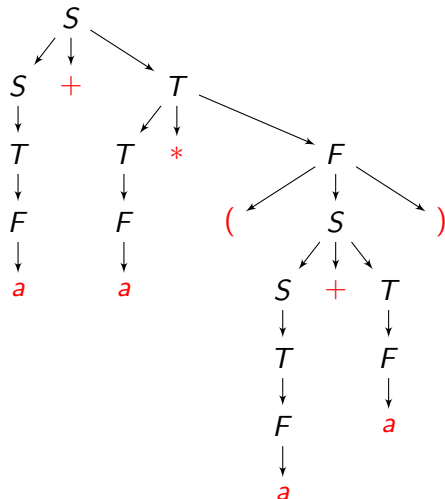
$$S \rightarrow S + T \mid T$$

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

$$F \rightarrow (S) \mid a$$

Arbres de dérivation

Quelques expressions arithmétiques :



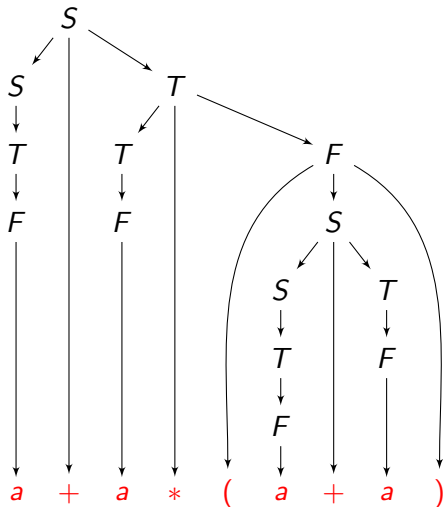
$$S \rightarrow S + T \mid T$$

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

$$F \rightarrow (S) \mid a$$

Arbres de dérivation

Quelques expressions arithmétiques :



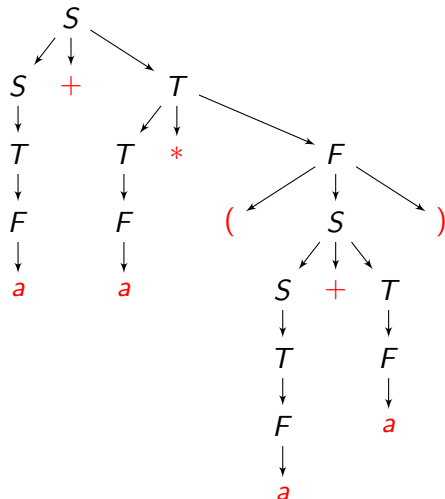
$$S \rightarrow S + T \mid T$$

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

$$F \rightarrow (S) \mid a$$

Arbres de dérivation

Quelques expressions arithmétiques :



$$S \rightarrow S + T \mid T$$

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

$$F \rightarrow (S) \mid a$$

- 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 $S \Rightarrow \alpha_1 \Rightarrow \dots \Rightarrow w$ 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 $S \Rightarrow^ w$.*

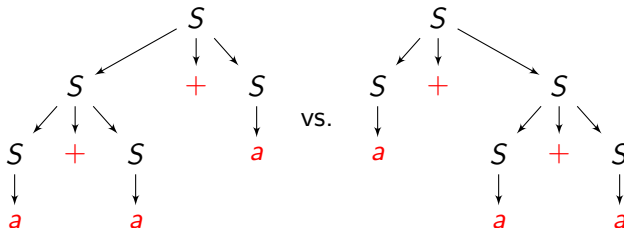
Ambiguïté

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

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 :

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

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

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

- overflow !

Ambiguïté

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.

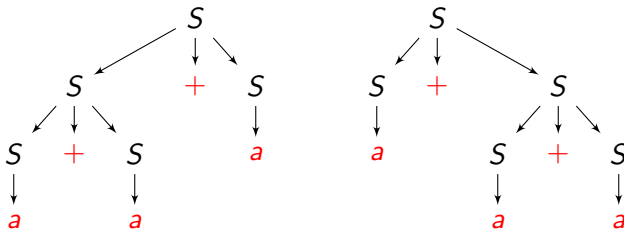
- par exemple $L = \{a^m b^n c^p \mid m = n \text{ ou } n = p\}$
- un langage **intrinsèquement ambigu**
- on ne peut pas traiter des langages intrinsèquement ambigus

Fin à l'ambiguïté

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

Exemple :

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



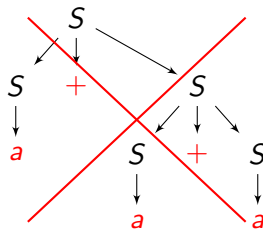
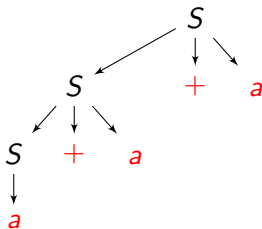
- $S \rightarrow S + T \mid T; T \rightarrow T * F \mid F; F \rightarrow (S) \mid a$: même chose

Fin à l'ambiguïté

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

Exemple :

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



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

Parsage LL(1)

Parsage

Problème de parsage

Pour une grammaire hc G , construire un algorithme qui :

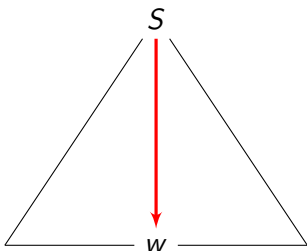
- pour un mot w , décide si $w \in L(G)$
- et dans le cas $w \in L(G)$, retourne l'arbre de dérivation

- arbre de dérivation de $w \triangleq$ sémantique de w

Nos algorithmes de parsage devrait

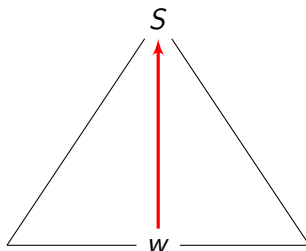
- pouvoir traiter des grammaires non-ambiguës
- avoir une complexité linéaire en taille d'entrée
- lire w de gauche à droite sans retour arrière

Approches



descendante

↑
maintenant



ascendante

↑
plus tard

Exemple

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$
$$\quad | \text{echo} \quad (2)$$
$$E \rightarrow \text{true} \quad (3)$$
$$\quad | \text{false} \quad (4)$$

Mot d'entrée :

if true then if false then echo fi fi

Arbre construit :

S

Exemple

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$
$$\quad | \text{echo} \quad (2)$$
$$E \rightarrow \text{true} \quad (3)$$
$$\quad | \text{false} \quad (4)$$

Mot d'entrée :

if true then if false then echo fi fi

Arbre construit :

S

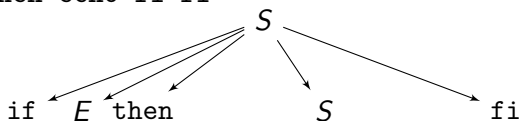
Exemple

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$
$$\quad \quad \quad | \text{ echo} \quad (2)$$
$$E \rightarrow \text{true} \quad (3)$$
$$\quad \quad \quad | \text{ false} \quad (4)$$

Mot d'entrée :

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

Arbre construit :



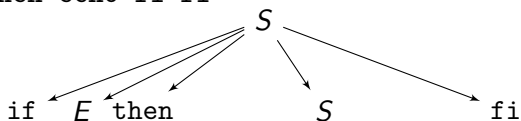
Exemple

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$
$$\quad \quad \quad | \text{ echo} \quad (2)$$
$$E \rightarrow \text{true} \quad (3)$$
$$\quad \quad \quad | \text{ false} \quad (4)$$

Mot d'entrée :

if **true** then if false then echo fi fi
(3)

Arbre construit :



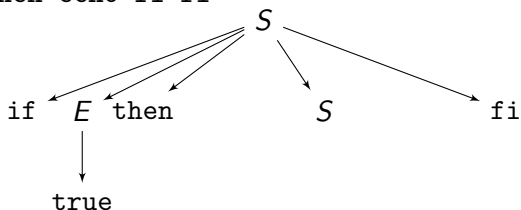
Exemple

 $S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$
 $\quad \quad \quad | \text{ echo} \quad (2)$
 $E \rightarrow \text{true} \quad (3)$
 $\quad \quad \quad | \text{ false} \quad (4)$

Mot d'entrée :

if **true** then if false then echo fi fi
(3)

Arbre construit :



Exemple

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$

$$| \text{echo} \quad (2)$$

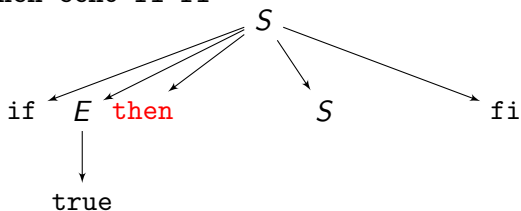
$$E \rightarrow \text{true} \quad (3)$$

$$| \text{false} \quad (4)$$

Mot d'entrée :

if true then if false then echo fi fi

Arbre construit :



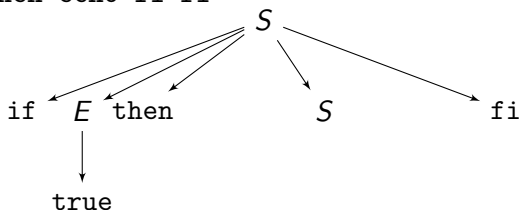
Exemple

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$
$$\quad \quad \quad | \text{ echo} \quad (2)$$
$$E \rightarrow \text{true} \quad (3)$$
$$\quad \quad \quad | \text{ false} \quad (4)$$

Mot d'entrée :

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

Arbre construit :



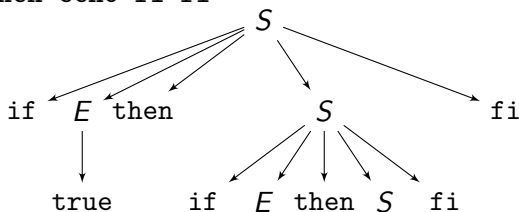
Exemple

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$
$$\quad \quad \quad | \text{ echo} \quad (2)$$
$$E \rightarrow \text{true} \quad (3)$$
$$\quad \quad \quad | \text{ false} \quad (4)$$

Mot d'entrée :

Arbre construit :

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



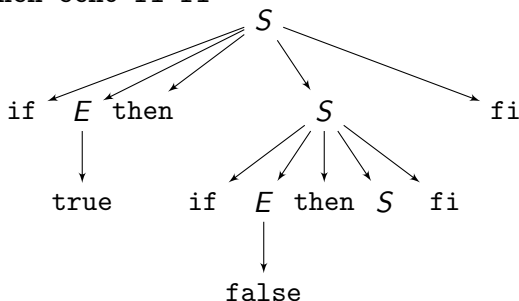
Exemple

 $S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$ $\quad \quad \quad | \text{ echo} \quad (2)$ $E \rightarrow \text{true} \quad (3)$ $\quad \quad \quad | \text{ false} \quad (4)$

Mot d'entrée :

Arbre construit :

if true then if false then echo fi fi
(4)



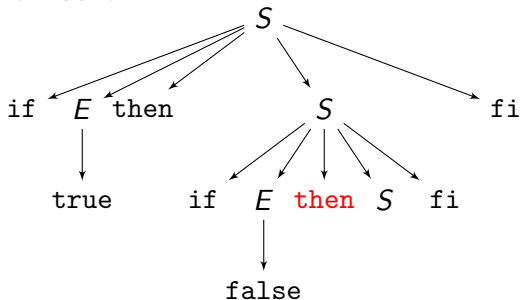
Exemple

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$
$$\quad \quad \quad | \text{ echo} \quad (2)$$
$$E \rightarrow \text{true} \quad (3)$$
$$\quad \quad \quad | \text{ false} \quad (4)$$

Mot d'entrée :

Arbre construit :

if true then if false then echo fi fi



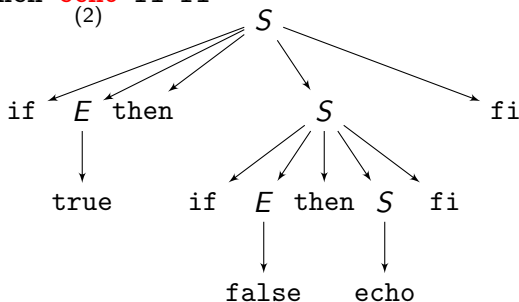
Exemple

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$
$$| \text{echo} \quad (2)$$
$$E \rightarrow \text{true} \quad (3)$$
$$| \text{false} \quad (4)$$

Mot d'entrée :

Arbre construit :

if true then if false then **echo** fi fi
(2)



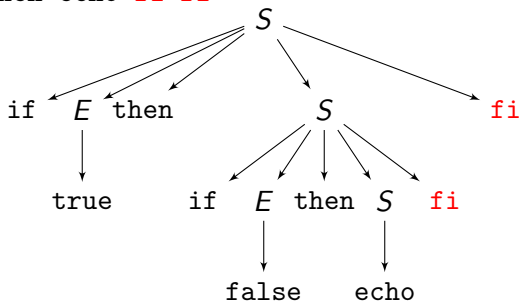
Exemple

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$
$$| \text{echo} \quad (2)$$
$$E \rightarrow \text{true} \quad (3)$$
$$| \text{false} \quad (4)$$

Mot d'entrée :

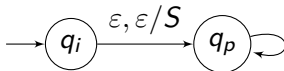
Arbre construit :

if true then if false then echo fi fi



Exemple, version Greibach

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \mid \text{echo}$$

$$E \rightarrow \text{true} \mid \text{false}$$


$$\text{if}, S/E \text{ then } S \text{ fi}$$

$$\text{echo}, S/\varepsilon$$

$$\text{true}, E/\varepsilon$$

$$\text{false}, E/\varepsilon$$

$$\text{then}, \text{then}/\varepsilon$$

$$\text{fi}, \text{fi}/\varepsilon$$

- 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	$E \text{ then } S \text{ fi}$	true then if false then echo fi fi
q_p	then $S \text{ fi}$	then if false then echo fi fi
q_p	$S \text{ fi}$	if false then echo fi fi
q_p	$E \text{ then } S \text{ fi fi}$	false then echo fi fi
q_p	then $S \text{ fi fi}$	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

 $S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$ $\quad \quad \quad | \text{ echo} \quad \quad \quad (2)$ $E \rightarrow \text{true} \quad (3)$ $\quad \quad \quad | \text{ false} \quad (4)$

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 $Z \rightarrow S\$$

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 $Z \rightarrow S\$$

Exemple :

$$\begin{array}{ll} S \rightarrow \text{if } E \text{ then } S \text{ fi} & (1) \\ | \text{echo} & (2) \end{array}$$
$$\begin{array}{ll} E \rightarrow \text{true} & (3) \\ | \text{false} & (4) \end{array}$$

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 $Z \rightarrow S\$$

Exemple :

$$Z \rightarrow S\$ \quad (0)$$

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$

$$\quad | \text{echo} \quad (2)$$

$$E \rightarrow \text{true} \quad (3)$$

$$\quad | \text{false} \quad (4)$$

FIRST

$$Z \rightarrow S\$ \quad (0)$$

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$

$$| \text{echo} \quad (2)$$

$$E \rightarrow \text{true} \quad (3)$$

$$| \text{false} \quad (4)$$

Pour construire la table de parsage, on a besoin de savoir quels terminaux peuvent apparaître à gauche d'une dérivation depuis une variable.

Définition (8.2)

Soit $A \in N$, alors $\text{FIRST}(A) \subseteq \Sigma$ est défini 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

$$Z \rightarrow S\$ \quad (0)$$

$$S \rightarrow \text{if } E \text{ then } S \text{ fi} \quad (1)$$

$$| \text{echo} \quad (2)$$

$$E \rightarrow \text{true} \quad (3)$$

$$| \text{false} \quad (4)$$

Pour construire la table de parsing, on a besoin de savoir quels terminaux peuvent apparaître à gauche d'une dérivation depuis une variable.

Définition (8.2)

Soit $A \in N$, alors $\text{FIRST}(A) \subseteq \Sigma$ est défini 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	if, echo
E	true, false

FIRST

$$Z \rightarrow S\$ \quad (0)$$

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

$$\quad | \text{echo} \quad (2) \qquad \quad | \text{false} \quad (4)$$

Pour construire la table de parcage, on a besoin de savoir quels terminaux peuvent apparaître à gauche d'une dérivation depuis une variable.

Définition (8.2)

Soit $A \in N$, alors $\text{FIRST}(A) \subseteq \Sigma$ est défini 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	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 $A \rightarrow Aw$?
 - 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 $A \rightarrow Aw$?
 - récursion à gauche : on ne l'aime pas, faut éviter
- ou si $A \rightarrow Bw$ 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 :

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

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

Définition

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

Exemple

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

$Z \rightarrow S\$$ (0)

$S \rightarrow \text{if } E \text{ then } S \text{ fi}$ (1)

 | echo (2)

$E \rightarrow \text{true}$ (3)

 | false (4)

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

$Z \rightarrow S\$$	(0)	$S \rightarrow \text{echo}$	(3)
$S \rightarrow \text{if } E \text{ then } S$	(1)	$E \rightarrow \text{true}$	(4)
$\text{if } E \text{ then } S \text{ else } S$	(2)	false	(5)

	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 $A \rightarrow X_1 \dots X_k \mid Y_1 \dots Y_\ell$ de productions, $X_1 \neq Y_1$.

Exemple :

$$\begin{aligned} S &\rightarrow \text{if } E \text{ then } S \\ &\quad \mid \text{if } E \text{ then } S \text{ else } S \end{aligned}$$

devient

$$\begin{aligned} S &\rightarrow \text{if } E \text{ then } SX \\ X &\rightarrow \text{else } S \mid \varepsilon \end{aligned}$$

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

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 $S \rightarrow \varepsilon$ ou $A \rightarrow a\alpha$ avec $a \in \Sigma$ et $\alpha \in (N \setminus \{S\})^*$.

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

Exemple :

$$\begin{array}{ccc}
 X \rightarrow Xu & \text{devient} & X \rightarrow vY \\
 | \ v & & Y \rightarrow uY \\
 & & | \ \varepsilon
 \end{array}$$

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

NULL

Définition (8.3)

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

```
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

$$Z \rightarrow XYZ \mid c$$

$$X \rightarrow Y \mid a$$

$$Y \rightarrow b \mid \varepsilon$$

NULL =

Exemple

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

FIRST avec ε , bis

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

```
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

$$Z \rightarrow XYZ \mid c$$

$$X \rightarrow Y \mid a$$

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

$$Y \rightarrow b \mid \varepsilon$$

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

Exemple, bis

$$Z \rightarrow XYZ \mid c$$

$$X \rightarrow Y \mid a \quad \text{NULL} = \{X, Y\}$$

$$Y \rightarrow b \mid \varepsilon$$

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

Exemple, bis

$$Z \rightarrow XYZ \mid c$$

$$X \rightarrow Y \mid a \quad \text{NULL} = \{X, Y\}$$

$$Y \rightarrow b \mid \varepsilon$$

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

Exemple, bis

$$Z \rightarrow XYZ \mid c$$

$$X \rightarrow Y \mid a$$

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

$$Y \rightarrow b \mid \varepsilon$$

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

Exemple, bis

$$Z \rightarrow XYZ \mid c$$

$$X \rightarrow Y \mid a$$

$$Y \rightarrow b \mid \varepsilon$$

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

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

A	FOLLOW(A)
X	
Y	
Z	

Exemple, bis

$$Z \rightarrow XYZ \mid c$$

$$X \rightarrow Y \mid a$$

$$Y \rightarrow b \mid \varepsilon$$

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

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 $X \rightarrow w \quad (n)$:
 - 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

$$Z \rightarrow XYZ \quad (1)$$

$$| c \quad (2)$$

$$X \rightarrow a \quad (3)$$

$$| Y \quad (4)$$

$$Y \rightarrow b \quad (5)$$

$$| \varepsilon \quad (6)$$

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

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

$$Z \rightarrow XYZ \quad (1)$$

$$\quad | c \quad (2)$$

$$X \rightarrow a \quad (3)$$

$$\quad | Y \quad (4)$$

$$Y \rightarrow b \quad (5)$$

$$\quad | \varepsilon \quad (6)$$

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

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

$$Z \rightarrow XYZ \quad (1)$$

$$\quad | c \quad (2)$$

$$X \rightarrow a \quad (3)$$

$$\quad | Y \quad (4)$$

$$Y \rightarrow b \quad (5)$$

$$\quad | \varepsilon \quad (6)$$

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

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

$$Z \rightarrow XYZ \quad (1)$$

$$| c \quad (2)$$

$$X \rightarrow a \quad (3)$$

$$| Y \quad (4)$$

$$Y \rightarrow b \quad (5)$$

$$| \varepsilon \quad (6)$$

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

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

$$Z \rightarrow XYZ \quad (1)$$

$$\quad | c \quad (2)$$

$$X \rightarrow a \quad (3)$$

$$\quad | Y \quad (4)$$

$$Y \rightarrow b \quad (5)$$

$$\quad | \varepsilon \quad (6)$$

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

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

$$Z \rightarrow XYZ \quad (1)$$

$$| c \quad (2)$$

$$X \rightarrow a \quad (3)$$

$$| Y \quad (4)$$

$$Y \rightarrow b \quad (5)$$

$$| \varepsilon \quad (6)$$

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

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

The image features a classic target graphic with a dark blue center and several concentric red rings of varying shades. The text "That's all Folks!" is written in a white, elegant cursive script, slanted diagonally across the center of the target.

That's all Folks!