Analyse Syntaxique

- On a vu les outils de l'analyse lexicale :
  - type de langages : réguliers ;
  - description : expressions régulières ;
  - formalisme sous-jacent : AF(N)D.
- Ces outils ne suffisent pas pour l'analyse syntaxique :
  - type de langages : algébriques ;
  - description : grammaires algébriques ;
  - formalisme sous-jacent : automate a pile (cours ultérieur).

Grammaire algébrique : Définition

- Une grammaire algébrique ou hors-contexte est un quadruplet
- $G = (V_T, V_N, P, S)$
- V_T et V_N sont des vocabulaires disjoints $V_T \cap V_N = \emptyset$:
  - V_T est l'ensemble des terminaux,
  - V_N l'ensemble des non-terminaux ;
- $P \subseteq V_N \times (V_N \cup V_T)^*$ est l'ensemble des (règles de) productions
- $S \in V_N$ est l'axiome, ou symbole de départ.

Grammaire algébrique : Définition

$P \subseteq V_N \times (V_N \cup V_T)^*$

- Une règle de production $\alpha \rightarrow \beta$ indique que la séquence de symboles $\alpha \in (V_N)$ peut être remplacée par la séquence de symboles $\in (V_N \cup V_T)^*$
- α est appelé partie gauche de la production β est appelé partie droite

Conventions :

- Majuscules non-terminaux
- Minuscule du début de l'alphabet terminaux
- Minuscule de la fin de l'alphabet chaîne de terminaux
- Lettres grecques chaîne de terminaux et non-terminaux

Grammaire algébrique : Exemple

- $G = (V_T, V_N, P, S)$ $\qquad$ $P \subseteq V_N \times (V_N \cup V_T)^*$
- V_T = { il, elle ,parle, devient, est, court, sympathique}
- V_N = { PHRASE, PRONOM, VERBE, COMPLEMENT, VERBEETAT, VERBEACTION }
- P = { PHRASE → PRONOM VERBE COMPLEMENT,
  PRONOM → il | elle
  VERBE→ VERBEETAT | VERBEACTION
  VERBEETAT → est | devient | reste
  VERBEACTION → parle | court
  COMPLEMENT → vite | sympathique
- S = PHRASE

Grammaire algébrique : Remarques

- Plutôt que d'écrire plusieurs productions du type $\alpha \rightarrow \beta$ et $\alpha \rightarrow \gamma$ on écrit $\alpha \rightarrow \beta \mid \gamma$. Notations de type BNF (Backus-Naur Form) et EBNF (Extended BNF) qui introduit en plus des [ ] pour les éléments optionnels et * ? …
- L'application de plusieurs productions successive s'appelle dérivation $\alpha \rightarrow \beta$ puis $\beta \rightarrow \gamma$ peut se traduire $\alpha \rightarrow^2 \gamma$ une dérivation avec 2 productions
  $\alpha \rightarrow^* \gamma$ une dérivation de plusieurs productions
- On remarquera que « il court vite » est un mot de 3 lettres pour l'analyseur syntaxique ce sont les 3 lexèmes fourni par l'analyseur lexical pour lequel, il s'agit de 3 mots de 2, 5 et 4 lettres
  la désignation mot/lettre est donc tout à fait relative

Grammaire algébrique : Exemple

- « il court vite » est-il syntaxiquement correct ? correspond-il à la grammaire précédente ?
- On part de l’axiome :
  S = PHRASE
  → PRONOM VERBE COMPLEMENT
  → PRONOM VERBEACTION COMPLEMENT
  → il VERBEACTION COMPLEMENT
  → il court COMPLEMENT
  → Il court vite
- En partant de l’axiome, on arrive, en appliquant les bonnes productions à la phrase testée
- Lorsqu’on procède dans ce sens, on parle d’analyse descendante

Hiérarchie de Chomsky

Une grammaire est dite :

- de type 3 ou **rationnelle** si chaque production est de la forme **A** → **wB** ou **A** → **w** où **A** et **B** sont des non terminaux et **w** est un terminal ;
- de type 2 ou **algébrique** ou **non contextuelle** si chaque production est de la forme **A** →α ou **A** est un non terminal et α est une séquence de terminaux ou non terminaux ;
- de type 1 ou **contextuelle** si chaque production est de la forme α**A**$\beta \rightarrow \alpha\ \gamma\ \beta$ ou α, β et γ sont des séquences de terminaux ou non terminaux et $\gamma \neq \varepsilon$;
- de type 0 ou **générale** si chaque production est de la forme $\alpha \rightarrow \beta$ sans autre contrainte.

Grammaire algébrique : Définition

- On note L(G) le langage généré par la grammaire G et défini par $\{w \in (V_T)^* \text{ tq } S \rightarrow^* w\}$
- On appelle arbre de dérivation (ou arbre syntaxique) un arbre tel que :
  - La racine est l'axiome
  - Les feuilles sont les symboles terminaux
  - Les nœuds sont les symboles non-terminaux
  - Les fils d'un nœud X sont α_0 ,..., α_n ssi $X \rightarrow \alpha_0 \ldots \alpha_n$ est une production [$\alpha_i \in (V_N \cup V_T)$]

Grammaire algébrique : Exemple

$G = (V_T, V_N, P, S)$ avec

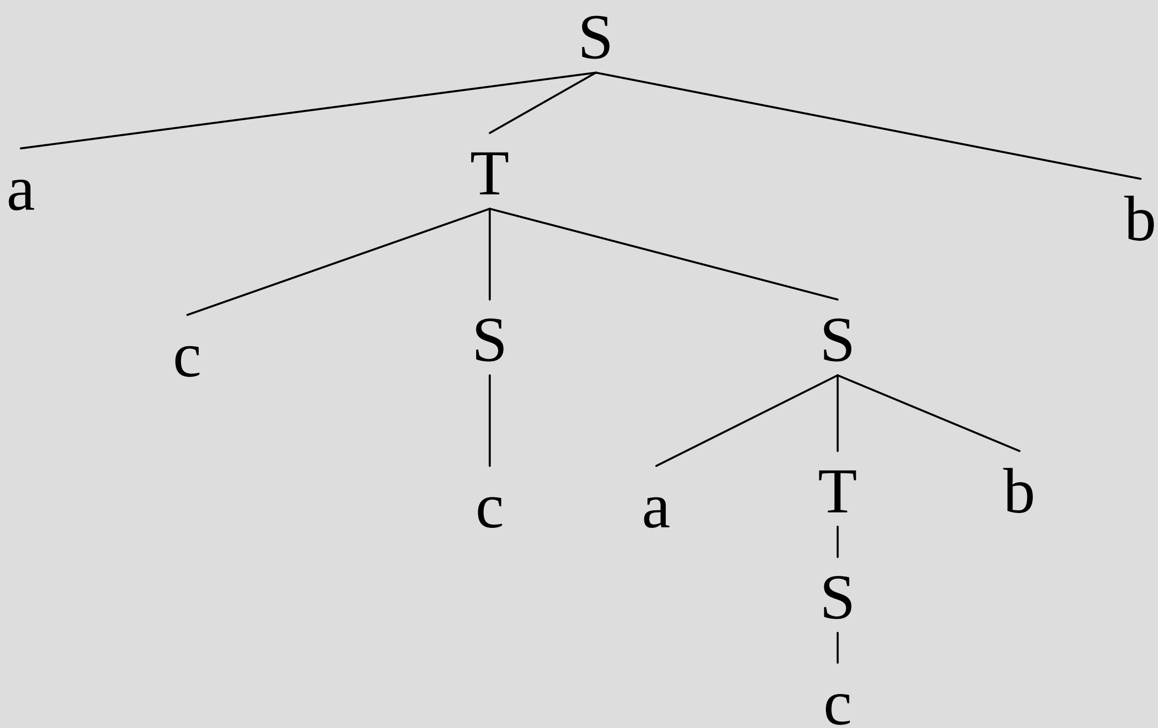
- $V_T = \{a,b\}$
- $V_N = \{S\}$
- Axiome S
- $P : S \rightarrow \varepsilon \mid a\,S\,b$

$S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaSbbb \rightarrow aaa\varepsilon bbb = aaabbb$

$L(G) = \{a^n b^n, n \geq 0\}$

Arbre de dérivation

- $G=(\{a,b,c\},\{S,T\},(S \to aTb|c, T \to cSS|S),S)$
- Arbre de dérivation pour accacbb



$S \to aTb \to acSSb$

$\to accSb \to accaTbb \to accaSbb \to accacbb$

$\to acSaTbb \to acSaSbb \to acSacbb \to accacbb$

Grammaire algébrique : ambiguë

- Les 2 dérivations (à gauche et à droite) précédentes conduisent au même arbre.
- Ce n’est pas toujours le cas:
  On dit que la grammaire G est ambiguë s’il existe un mot de L(G) qui a plusieurs arbres syntaxique
- Prenons l’exemple de l’INSTRUCTION conditionnelle de java, c, pascal …
- Elle existe sous 2 formes : avec ou sans partie sinon

if TEST INSTRUCTION

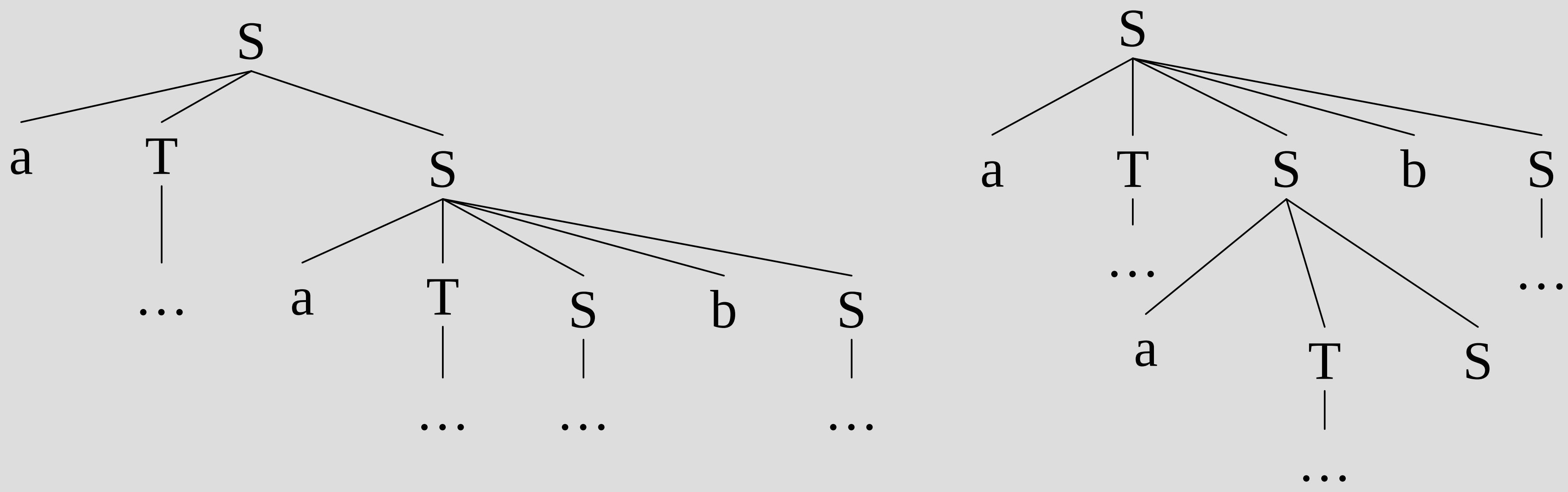
if TEST INSTRUCTION else INSTRUCTION

Grammaire algébrique : ambiguë

- Un extrait de la grammaire G
- V_T = {if, else}={a,b}
- V_N = {INSTRUCTION, TEST}={S,T}
- $S \rightarrow a\ T\ S \mid a\ T\ S\ b\ S \mid \ldots$
- $T \rightarrow \ldots$
- L'instruction :
  if test1 if test2 instruction1 else instruction2
  correspond-elle à $S \rightarrow aTS \rightarrow aTaTSbS$
  ou $S \rightarrow aTSbS \rightarrow aTaTSbS$
  autrement dit le else correspond-il au premier ou au second if?

Grammaire algébrique : ambiguë

- S → aTS → aTaTSbS   S → aTSbS → aTaTSbS

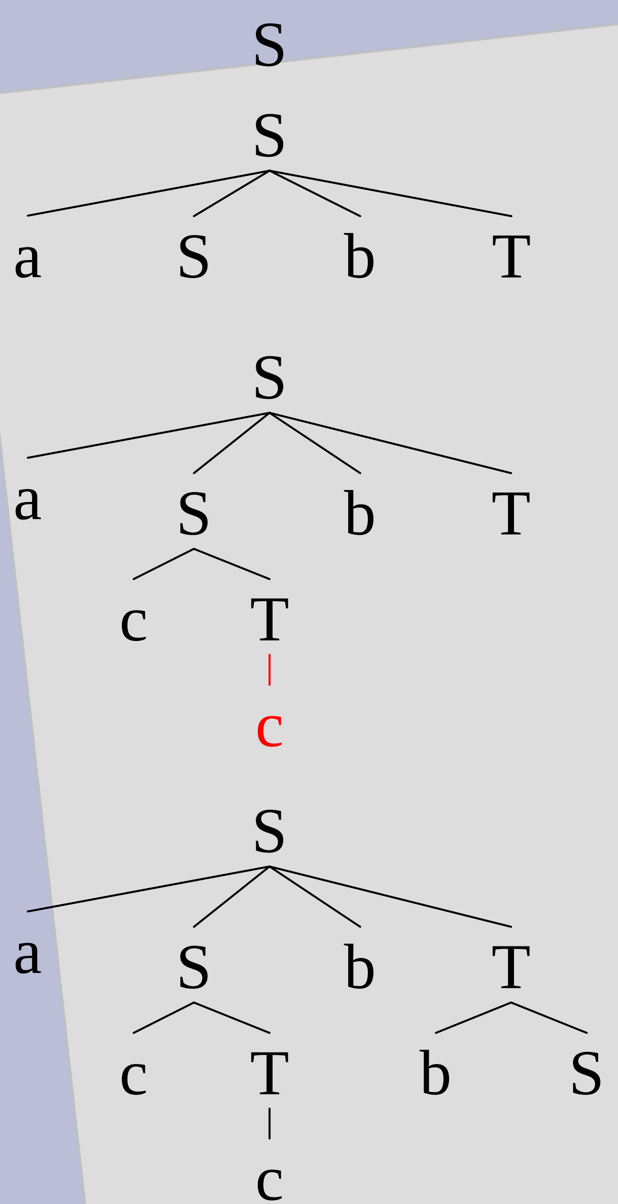


Grammaire algébrique : Analyse descendante

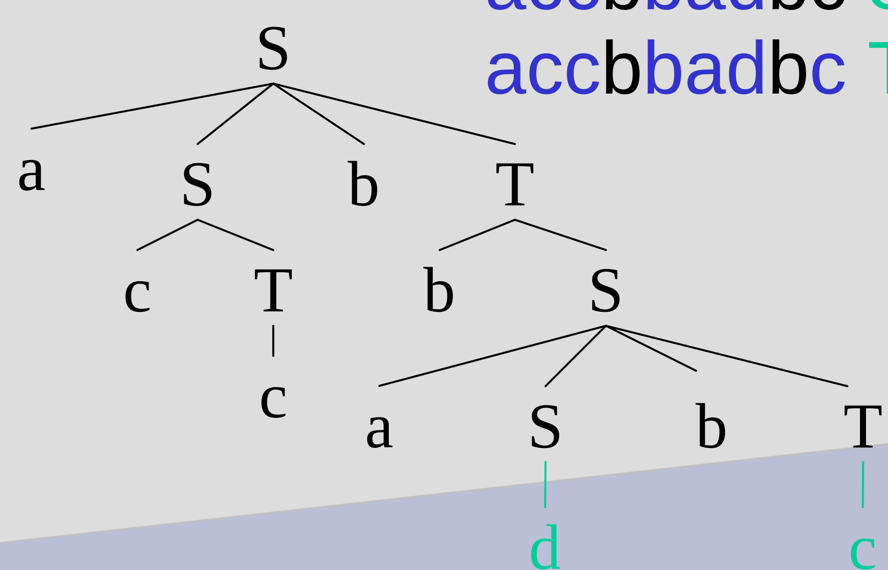
- Principe : construire l'arbre en partant de la racine (l'axiome) vers les feuilles (les unités lexicales)
- Méthode : on place l'axiome à la racine puis on lit les lettres du mot à tester et on avance pas à pas dans la construction
- Exemple :
- Le mot accbbadbc appartient-il au langage défini par la grammaire d'axiome S

$$\begin{cases} S \rightarrow aSbT \mid cT \mid d \\ T \rightarrow aT \mid bS \mid c \end{cases}$$

Analyse descendante : Exemple 1

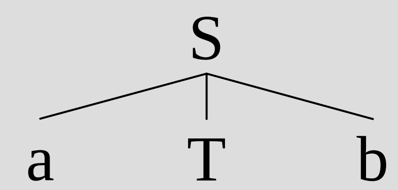


- $\begin{cases} S \rightarrow aSbT \mid cT \mid d \\ T \rightarrow aT \mid bS \mid c \end{cases}$
- accbbadbc $S \rightarrow aSbT$
  accbbadbc $S \rightarrow cT$
  accbbadbc $T \rightarrow c$
  accbbadbc $T \rightarrow bS$
  accbbadbc $S \rightarrow aSbT$
  accbbadbc $S \rightarrow d$
  accbbadbc $T \rightarrow c$



Analyse descendante : Exemple 2

- $\begin{cases} S \rightarrow aTb \\ T \rightarrow cd \mid c \end{cases}$
- acb appartient-il au langage?
- acb $S \rightarrow aTb$
  acb $T \rightarrow cd$ ou $T \rightarrow c$??
  Il faut lire la lettre suivante
  pour savoir quelle règle
  appliquer !



Analyse descendante : Exemple 3

- $S \rightarrow aSb \mid aSc \mid d$ et w = aaaaaaadbbcbbbc
- Dans cet exemple il faut lire la dernière lettre pour savoir qu'il faut appliquer $S \rightarrow aSc$
- C'est facile lorsqu'il y a peu de règle et peu de choix par règles
- On va donc construire une table appelée table d'analyse LL(1) qui permettra de savoir, avec certitude, quelle règle appliquer lorsque l'on lit un symbole
- Pour cela, on va construire 2 ensembles : PREMIER et SUIVANT

Table d'analyse LL(1) : Calcul de PREMIER

PREMIER(α)={a $\in V_T$ tq $\alpha \rightarrow^* a\, \beta$ où $\alpha, \beta \in V_N \cup V_T$ }

$\varepsilon \in$ PREMIER(α) $\Leftrightarrow \alpha \rightarrow^* \varepsilon$

PREMIER(α) est l'ensemble des terminaux qui peuvent commencer une chaîne qui dérive de α

Table d'analyse LL(1) : Exemple de PREMIER

- $\begin{cases} S \rightarrow Ta \\ T \rightarrow cU \mid bU \mid U \mid \varepsilon \\ U \rightarrow dS \end{cases}$
- $S \rightarrow Ta \rightarrow \varepsilon a = a$
  $S \rightarrow Ta \rightarrow bUa$
  $S \rightarrow Ta \rightarrow cUa$
  $S \rightarrow Ta \rightarrow Ua \rightarrow dSa$
- Premier(S)={a,b,c,d}
- Premier(T)={ε,b,c,d}
- Premier(aT)={a}
- Premier(TS)={a,b,c,d} $TS \rightarrow \varepsilon S \rightarrow Ta \rightarrow \varepsilon a = a$

Table d'analyse LL(1) : Algorithme de PREMIER

Si $\alpha \in V_N$ et $\alpha \rightarrow \alpha_1\ \alpha_2 \ldots \alpha_n$ une production où

$\alpha_i \in V_N \cup V_T \Rightarrow$ PREMIER(α_1)-{ε} $\subseteq$ PREMIER(α)

- Si $\exists$ j $\in$[2..n] tq $\forall$ i $\in$[1..j-1] et ε $\in$ PREMIER(α_i)
  $\Rightarrow$ PREMIER(α_j)-{ε} $\subseteq$ PREMIER(α)
- Si ε $\in$ PREMIER(α_j) $\forall$ i $\in$[1..n]
  $\Rightarrow$ ε $\in$ PREMIER(α)
- Si $\alpha \rightarrow$ ε est une production $\Rightarrow$ ε $\in$ PREMIER(α)

Si $\alpha \in V_T \Rightarrow$ PREMIER(α) = α

Table d'analyse LL(1) : Algorithme de PREMIER

Autre formulation :

- Si X est un terminal **alors** Premier(X) est juste X!
- S'il y a une production $X \rightarrow \varepsilon$ **alors** ajouter ε à Premier(X)
- S'il y a une production $X \rightarrow \alpha_1\ \alpha_2 \ldots \alpha_n$ **alors** ajoutez Premier($\alpha_1\ \alpha_2 \ldots \alpha_n$) à Premier(X)
- Premier($\alpha_1\ \alpha_2 \ldots \alpha_n$) est **soit**
  - Premier(α_1) (si Premier(α_1) ne contient pas de ε)
  - **Sinon** (si Premier(α_1) contient ε)
    Premier($\alpha_1\ \alpha_2 \ldots \alpha_n$) = (Premier($\alpha_1$) – $\{\varepsilon\}$) $\cup$ Premier($\alpha_2 \ldots \alpha_n$)
  - Si Premier(α_1) Premier(α_2) .. Premier(α_n) contiennent tous ε alors ε appartient à Premier($\alpha_1\ \alpha_2 \ldots \alpha_n$) aussi.

LL(1) Algorithme de PREMIER : application

Exemple 1

- $\begin{cases} S \rightarrow TUBe \\ T \rightarrow aT \mid \varepsilon \\ U \rightarrow bU \mid cU \mid \varepsilon \\ B \rightarrow de \mid da \mid dT \end{cases}$

- PREMIER(B) ={d}
  PREMIER(T) ={ε,a}
  PREMIER(U) ={ε,b,c}

- PREMIER(T)-{ε}={a} $\subseteq$ PREMIER(S)
  ε $\in$ PREMIER(T) $\Rightarrow$ PREMIER(U)-{ε}={b,c} $\subseteq$ PREMIER(S)
  ε $\in$ PREMIER(U) $\Rightarrow$ PREMIER(B)-{ε}={d} $\subseteq$ PREMIER(S)
  ε $\notin$ PREMIER(e)={e} et $\nexists$ S $\rightarrow$ ε $\Rightarrow$ ε $\notin$ PREMIER(S)

- Donc PREMIER(S) = {a,b,c,d}

Table d'analyse LL(1) : Calcul de SUIVANT

$\forall A \in V_T$ SUIVANT(A) = {a $\in V_T$ tq $S \rightarrow^* \alpha\ A\ a\ \beta$ où $\alpha, \beta \in V_N \cup V_T$ }

SUIVANT(A) est l'ensemble des terminaux qui peuvent apparaître à droite de A dans une dérivation de l'axiome S

Table d'analyse LL(1) : Exemple de Suivant

- $\begin{cases} S \to Sc \mid Ta \\ T \to TUa \mid bUb \mid U \mid \varepsilon \\ U \to dS \end{cases}$
- $S \to Sc$
  $S \to Ta \to Ua \to dSa$
  $S \to Ta \to bUba \to bdSba$
  $S \to Ta \to TUaa \to UUaa \to dSdSaa$
- SUIVANT(S)={a,b,c,d}
- $S \to Ta$; $S \to Ta \to TUaa \to TdSaa$
- $\{a,b\} \subseteq$ SUIVANT(T)
- $c \in$ SUIVANT(T) ? $d \in$ SUIVANT(T) ?
  →Trouver la dérivation qui va bien ???

Table d’analyse LL(1) : Algorithme de SUIVANT

1. Ajouter un marqueur de fin de chaîne (symbole $ par exemple) à SUIVANT(*S*) (où *S* est l'axiome de départ de la grammaire)
2. S'il y a une production $A \rightarrow \alpha B \beta$ où *B* est un non-terminal, alors ajouter le contenu de PREMIER(β) à SUIVANT(*B*), **sauf** ε
3. S'il y a une production $A \rightarrow \alpha B$, alors ajouter SUIVANT(*A*) à SUIVANT(*B*)
4. S'il y a une production $A \rightarrow \alpha B \beta$ avec $\varepsilon \in$ PREMIER (β) , alors ajouter SUIVANT(*A*) à SUIVANT(*B*)
5. Recommencer à partir de l'étape 3 jusqu'à ce qu'on n'ajoute rien de nouveau dans les ensembles SUIVANT.

LL(1) Algorithme de SUIVANT : application

- Exemple 1 :

$$\begin{cases} S \rightarrow aSb \mid cd \mid SAe \\ A \rightarrow aAdB \mid \varepsilon \\ B \rightarrow bb \end{cases}$$

- $S \rightarrow aSb$

PREMIER(b) $\subseteq$ SUIVANT(S) (2)

- $A \rightarrow aAdB$

PREMIER(d) $\subseteq$ SUIVANT(A) (2)

SUIVANT(A) $\subseteq$ SUIVANT(B) (3)

- $S \rightarrow SAe$

PREMIER(e) $\subseteq$ SUIVANT(A) (2)

PREMIER(A)-{ε} $\subseteq$ SUIVANT(S) (2)

	PREMIER	SUIVANT
S	*a c*	$ *b a*
A	*a* ε	*e d*
B	*b*	*e d*

LL(1) Algorithme de SUIVANT : application

- Exemple 2 : axiome E

$$\begin{cases} E \to TE' \\ E' \to +TE' \mid -TE' \mid \varepsilon \\ T \to FT' \\ T' \to *FT' \mid /FT' \mid \varepsilon \\ F \to (E) \mid nb \end{cases}$$

PREMIER(E) = PREMIER(T)

PREMIER(T) = PREMIER(F) ={(,nb}

- $E \to TE'$

PREMIER(E') ⊆ SUIVANT(T) (2)

SUIVANT(E) ⊆ SUIVANT(E') (3)

SUIVANT(E) ⊆ SUIVANT(T) (4)

- $T \to FT'$

PREMIER(T') ⊆ SUIVANT(F) (2)

SUIVANT(T) ⊆ SUIVANT(T') (3)

SUIVANT(T) ⊆ SUIVANT(F) (4)

	PREMIER	SUIVANT
E	*(nb*	*$)*
E'	+ - ε	*$)*
T	*(nb*	*$)* + -
T'	* / ε	*$)* + -
F	*(nb*	*$)* + - * /

Construction de la table d'analyse LL

- C'est un tableau T à 2 dimensions qui indique pour chaque $A \in V_N$ et chaque $a \in V_T$ ou \$ la règle de production à appliquer
- Pour chaque production $A \rightarrow \alpha$:
  $\forall a \neq \varepsilon$ et $a \in$ PREMIER(α) $\Rightarrow$ T[A,a] = $A \rightarrow \alpha$
  $\forall a \in$ SUIVANT(A) et $\varepsilon \in$ PREMIER(α) $\Rightarrow$ T[A,a] = $A \rightarrow \alpha$
- Chaque cellule vide T[A,a] est une erreur de syntaxe

Table d'analyse LL : application

Pour chaque $A \to \alpha$

$\forall a \neq \varepsilon$ et $a \in$ PREMIER$(\alpha) \Rightarrow T[A,a] = A \to \alpha$

$\forall a \in$ SUIVANT(A) et $\varepsilon \in$ PREMIER$(\alpha) \Rightarrow$

$T[A,a] = A \to \alpha$

$$
\begin{cases}
E \to TE' \\
E' \to +TE' \mid -TE' \mid \varepsilon \\
T \to FT' \\
T' \to *FT' \mid /FT' \mid \varepsilon \\
F \to (E) \mid nb
\end{cases}
$$

	PREMIER	SUIVANT
E	*(nb*	*$)*
E'	+ - ε	*$)*
T	*(nb*	*$)* + -
T'	* / ε	*$)* + -
F	*(nb*	*$)* + - * /

	nb	+	-	*	/	(	)	$
E	E→ TE'					E→ TE'		
E'		E' → +TE'	E' → -TE'				E' → ε	E' → ε
T	T → FT'					T → FT'		
T'		T' → ε	T' → ε	T' → *FT'	T' → /FT'		T' → ε	T' → ε
F	F → nb					F → (E)		

Table d'analyse LL : Utilisation/Algorithme

- Utilisation d'une pile initialisée avec \$ et l'axiome S
  et un pointeur p sur la première lettre du mot à analyser (terminé par \$)

- Répéter

```
    X le symbole au sommet de pile
    a la lettre pointée par p
    Si X ∈ V_N alors
            Si T[X,a]= X → α1 α2 …αn
                    dépiler X
                    empiler αn … α2 α1
            sinon ERREUR fsi
    sinon
            Si X=$ alors
                    Si A=$ alors ACCEPTER sinon ERREUR fsi
            Sinon
                    Si X=a alors
                            depiler X; avancer ps
                    Sinon ERREUR fsi
            fsi
    fsi
jusqu'à ERREUR ou ACCEPTER
```

Table d'analyse LL : Application

Pile	Entrée	Sortie
$E	3+4*5$	

	nb	+	-	*	/	(	)	$
E	E→ TE'					E→ TE'		
E'		E' → +TE'	E' → -TE'				E' → ε	E' → ε'
T	T → FT'					T → FT'		
T'		T' → ε	T' → ε	T' → *FT'	T' → /FT'		T' → ε	T' → ε
F	F → nb					F → (E)		

E

Table d’analyse LL : Application

Pile	Entrée	Sortie
$E	3+4*5$	E → TE’

	nb	+	-	*	/	(	)	$
E	E→ TE’					E→ TE’		
E’		E’ → +TE’	E’ → -TE’				E’ → ε	E’ → ε’
T	T → FT’					T → FT’		
T’		T’ → ε	T’ → ε	T’ → *FT’	T’ → /FT’		T’ → ε	T’ → ε
F	F → nb					F → (E)		

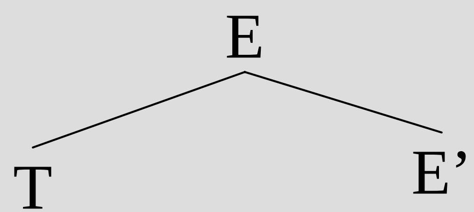


Table d’analyse LL : Application

Pile	Entrée	Sortie
$E	3+4*5$	E → TE’
$E’T	3+4*5$	

	nb	+	-	*	/	(	)	$
E	E→ TE’					E→ TE’		
E’		E’ → +TE’	E’ → -TE’				E’ → ε	E’ → ε’
T	T → FT’					T → FT’		
T’		T’ → ε	T’ → ε	T’ → *FT’	T’ → /FT’		T’ → ε	T’ → ε
F	F → nb					F → (E)		

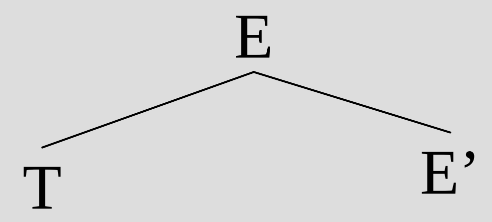


Table d'analyse LL : Application

Pile	Entrée	Sortie
$E	3+4*5$	E → TE'
$E'T	3+4*5$	T → FT'

	nb	+	-	*	/	(	)	$
E	E→ TE'					E→ TE'		
E'		E' → +TE'	E' → -TE'				E' → ε	E' → ε'
T	T → FT'					T → FT'		
T'		T' → ε	T' → ε	T' → *FT'	T' → /FT'		T' → ε	T' → ε
F	F → nb					F → (E)		

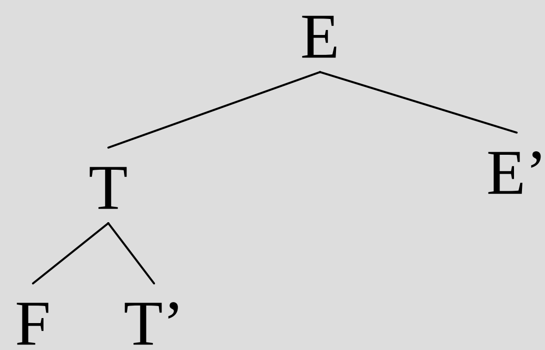


Table d’analyse LL : Application

Pile	Entrée	Sortie
$E	3+4*5$	E → TE’
$E’T	3+4*5$	T → FT’
$E’T’F	3+4*5$	F → nb
$E’T’3	3+4*5$	

	nb	+	-	*	/	(	)	$
E	E→ TE’					E→ TE’		
E’		E’ → +TE’	E’ → -TE’				E’ → ε	E’ → ε’
T	T → FT’					T → FT’		
T’		T’ → ε	T’ → ε	T’ → *FT’	T’ → /FT’		T’ → ε	T’ → ε
F	F → nb					F → (E)		

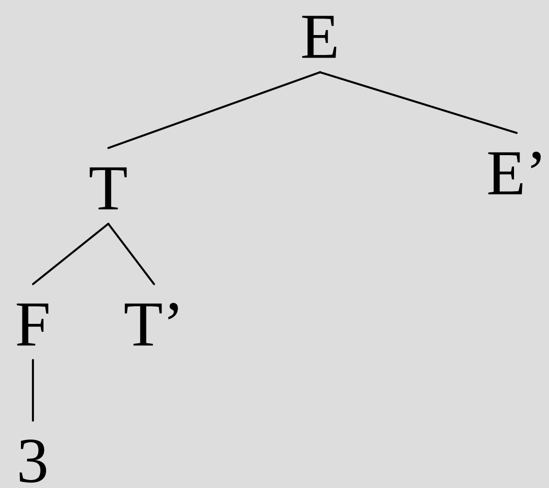


Table d'analyse LL : Application

Pile	Entrée	Sortie
$E	3+4*5$	E → TE'
$E'T	3+4*5$	T → FT'
$E'T'F	3+4*5$	F → nb
$E'T'3	3+4*5$	
$E'T'	+4*5$	T' → ε
$E'	+4*5$	E' → +TE'
$E'T+	+4*5$	

	nb	+	-	*	/	(	)	$
E	E→ TE'					E→ TE'		
E'		E' → +TE'	E' → -TE'				E' → ε	E' → ε'
T	T → FT'					T → FT'		
T'		T' → ε	T' → ε	T' → *FT'	T' → /FT'		T' → ε	T' → ε
F	F → nb					F → (E)		

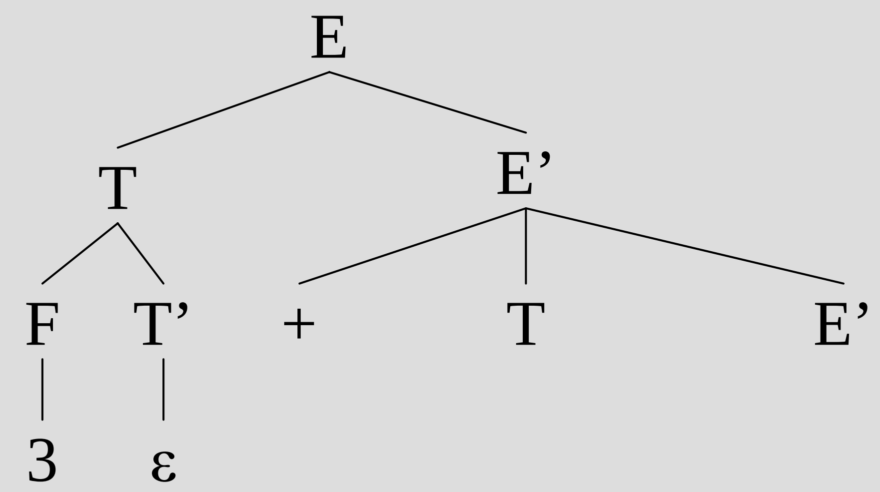


Table d’analyse LL : Application

Pile	Entrée	Sortie
$E	3+4*5$	E → TE’
$E’T	3+4*5$	T → FT’
$E’T’F	3+4*5$	F → nb
$E’T’3	3+4*5$	
$E’T’	+4*5$	T’ → ε
$E’	+4*5$	E’ → +TE’
$E’T+	+4*5$	
$E’T	4*5$	T → FT’
$E’T’F	4*5$	F → nb
$E’T’4	4*5$	
$E’T’	*5$	

	nb	+	-	*	/	(	)	$
E	E→ TE’					E→ TE’		
E’		E’ → +TE’	E’ → -TE’				E’ → ε	E’ → ε’
T	T → FT’					T → FT’		
T’		T’ → ε	T’ → ε	T’ → *FT’	T’ → /FT’		T’ → ε	T’ → ε
F	F → nb					F → (E)		

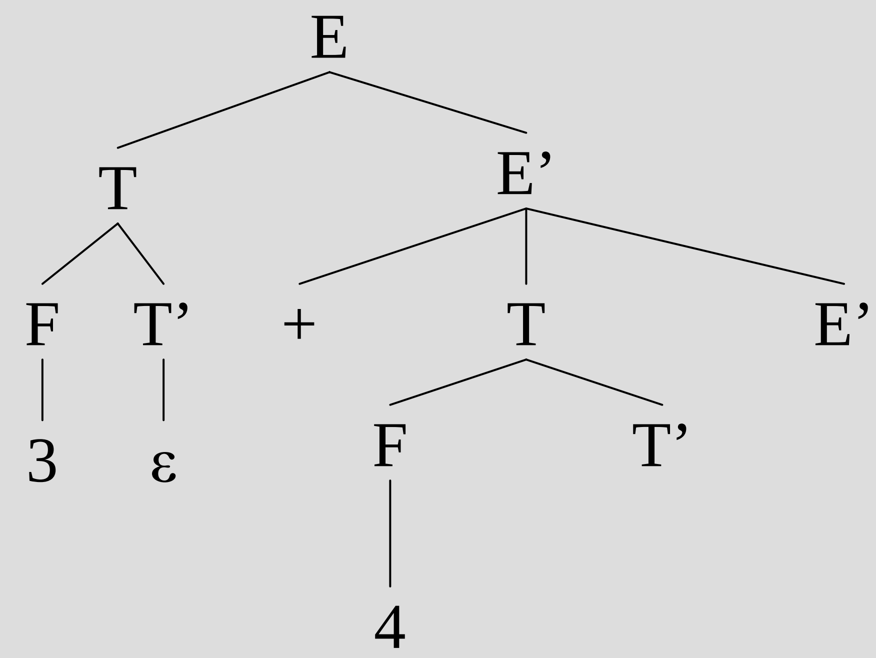


Table d'analyse LL : Application

Pile	Entrée	Sortie
$E	3+4*5$	E → TE'
$E'T	3+4*5$	T → FT'
$E'T'F	3+4*5$	F → nb
$E'T'3	3+4*5$	
$E'T'	+4*5$	T' → ε
$E'	+4*5$	E' → +TE'
$E'T+	+4*5$	
$E'T	4*5$	T → FT'
$E'T'F	4*5$	F → nb
$E'T'4	4*5$	
$E'T'	*5$	T' → *FT'
$E'T'F*	*5$	
$E'T'F	5$	F → nb
$E'T'5	5$	
$E'T'	$	

	nb	+	-	*	/	(	)	$
E	E→ TE'					E→ TE'		
E'		E' → +TE'	E' → -TE'				E' → ε	E' → ε'
T	T → FT'					T → FT'		
T'		T' → ε	T' → ε	T' → *FT'	T' → /FT'		T' → ε	T' → ε
F	F → nb					F → (E)		

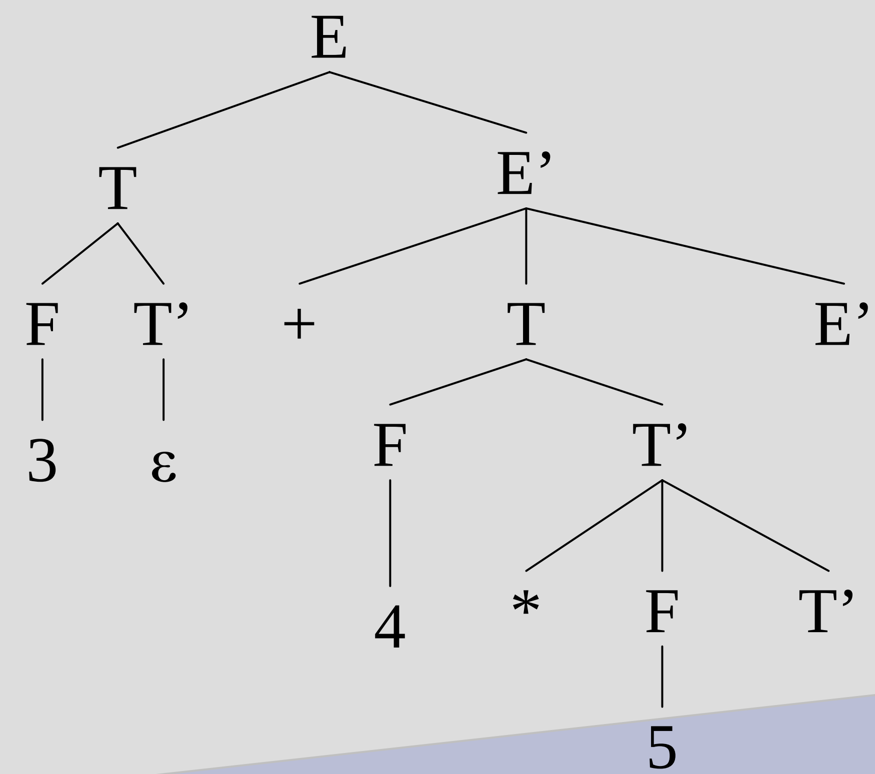
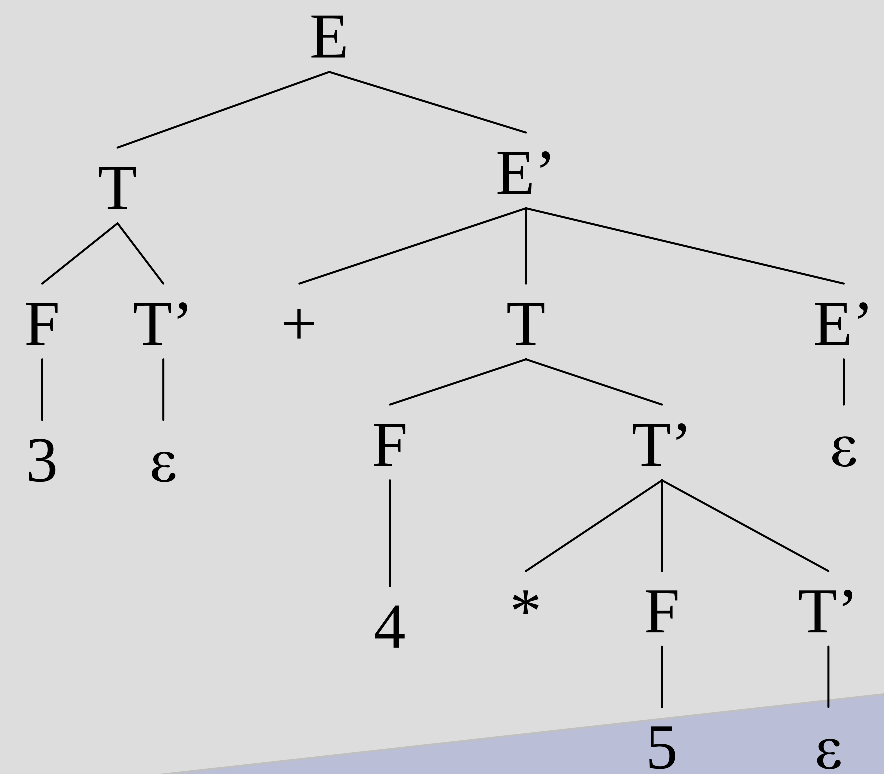


Table d’analyse LL : Application

Pile	Entrée	Sortie
$E	3+4*5$	E → TE’
$E’T	3+4*5$	T → FT’
…	…	…
…	…	…
$E’T	4*5$	T → FT’
$E’T’F	4*5$	F → nb
$E’T’4	4*5$	
$E’T’	*5$	T’ → *FT’
$E’T’F*	*5$	
$E’T’F	5$	F → nb
$E’T’5	5$	
$E’T’	$	
$E’T’	$	T’ → ε
$E’	$	E’ → ε
$	$	ACCEPTER

	nb	+	-	*	/	(	)	$
E	E→ TE’					E→ TE’		
E’		E’ → +TE’	E’ → -TE’				E’ → ε	E’ → ε’
T	T → FT’					T → FT’		
T’		T’ → ε	T’ → ε	T’ → *FT’	T’ → /FT’		T’ → ε	T’ → ε
F	F → nb					F → (E)		



Grammaire LL(1)

- S’il y a plusieurs productions dans les cellules de la table d’analyse, l’algorithme précédent ne pourra être appliqué :
  On appelle grammaire LL(1) une grammaire pour laquelle la table d’analyse contient au plus une production

- LL(1) signifie que l’on parcourt l’entrée de gauche (Left) à droite et que l’on utilise des dérivation gauches (Leftmost) et un seul symbole est nécessaire pour utiliser la table d’analyse

- Une grammaire <u>n’est pas</u> LL(1) lorsqu’elle est ambiguë ou récursive à gauche ou non factorisée à gauche.

Récursivité à gauche

- Définition :
une grammaire est immédiatement récursive à gauche si elle contient $A \in V_N$ tq $A \rightarrow A\alpha$
- Exemple:

$$\begin{cases} S \rightarrow SaT \mid U \\ T \rightarrow Tc \mid \varepsilon \\ U \rightarrow Ub \mid d \end{cases}$$

Récursivité à gauche immédiate : Élimination

- Remplacer toutes les règles de la forme: $A \rightarrow A\alpha|\beta$ par:

$$\begin{cases} A \rightarrow \beta A' \\ A' \rightarrow \alpha A' \mid \varepsilon \end{cases}$$

- Exemple:

$$\begin{cases} S \rightarrow SaT \mid U \\ T \rightarrow Tc \mid \varepsilon \\ U \rightarrow Ub \mid d \mid e \end{cases} \Rightarrow$$

- $$\begin{cases} S \rightarrow US' \\ S' \rightarrow aTS' \mid \varepsilon \\ T \rightarrow T' \\ T' \rightarrow cT' \mid \varepsilon \\ U \rightarrow dU' \mid eU' \\ U' \rightarrow bU' \mid \varepsilon \end{cases}$$

Ou

- $$\begin{cases} S \rightarrow US' \\ S' \rightarrow aTS' \mid \varepsilon \\ T \rightarrow cT \mid \varepsilon \\ U \rightarrow dU' \mid eU' \\ U' \rightarrow bU' \mid \varepsilon \end{cases}$$

Récursivité à gauche

- Définition :
  une grammaire est récursive à gauche si elle contient $A \in V_N$ tq $A \rightarrow^{+} A\alpha$
- Exemple:
  $$\begin{cases} S \rightarrow Ta \mid b \\ T \rightarrow Tc \mid Sd \mid c \end{cases}$$
- S est récursif à gauche mais pas immédiatement car
  $S \rightarrow Ta \rightarrow Sda$

Récursivité à gauche : Élimination

- Algorithme :
  - Ordonner les non-terminaux $A_1, A_2, \ldots A_n$
  - Pour i de 1 à n faire

    pour j de 1 à i-1 faire

    remplacer chaque production $A_i \rightarrow A_j\alpha \mid \gamma$ où

    $A_j \rightarrow \beta_1 \mid \ldots \mid \beta_p$ par $A_i \rightarrow \beta_1\alpha \mid \ldots \mid \beta_p \alpha \mid \gamma$

    fait

    éliminer les récursivités immédiates gauches des productions A_i

    fait

Récursivité à gauche : Exemple

RG : remplacer chaque production $Ai \to Aj\alpha \mid \gamma$
où $Aj \to \beta 1 \mid \ldots \mid \beta p$ par $Ai \to \beta 1\alpha \mid \ldots \mid \beta p\ \alpha \mid \gamma$

RGI : Remplacer toutes les règles de la forme:
$A \to A\alpha|\beta$ par: $A \to \beta\ A'$
$A' \to \alpha\ A' \mid \varepsilon$

- Exemple

$$\begin{cases} S \to Aa \mid b \\ A \to Ac \mid Sd \mid BA \mid c \\ B \to SSc \mid a \end{cases}$$

- On ordonne $A_1=S$, $A_2=A$, $A_3=B$
- i=1 pas de récursivité immédiate $S \to Aa \mid b$
- i=2, j=1 $A \to Ac \mid Aad \mid bd \mid BA \mid c$

  éliminer rec. imm. $A \to bdA' \mid BAA' \mid cA'$

  $A' \to cA' \mid adA' \mid \varepsilon$
- i=3, j=1 $B \to AaSc \mid bSc \mid a$
- i=3, j=2 $B \to bdA'aSc \mid BAA'aSc \mid cA'aSc \mid bSc \mid a$

Récursivité à gauche : Exemple (suite)

RGI : Remplacer toutes les règles de la forme: $B \rightarrow B\alpha|\beta$ par:
$B \rightarrow \beta B'$
$B' \rightarrow \alpha B' \mid \varepsilon$

- Exemple

$$\begin{cases} S \rightarrow Aa \mid b \\ A \rightarrow Ac \mid Sd \mid BA \mid c \\ B \rightarrow SSc \mid a \end{cases}$$

- $B \rightarrow bdA'aSc \mid B\overbrace{AA'aSc}^{\alpha} \mid cA'aSc \mid bSc \mid a$
- Élim. rec. imm. $B \rightarrow bdA'aScB' \mid cA'aScB' \mid bScB' \mid aB'$
  $B' \rightarrow AA'aScB' \mid \varepsilon$
- $$\begin{cases} S \rightarrow Aa \mid b \\ A \rightarrow bdA' \mid BAA' \mid cA' \\ A' \rightarrow cA' \mid adA' \mid \varepsilon \\ B \rightarrow bdA'aScB' \mid cA'aScB' \mid bScB' \mid aB' \\ B' \rightarrow AA'aScB' \mid \varepsilon \end{cases}$$

Grammaire propre

- Qui ne contient pas de production $A \rightarrow \varepsilon$
- Algorithme pour rendre une grammaire propre :
  En rajoutant, dans chaque production où A apparaît en partie droite et pour chaque A, une production où A est remplacé par ε
- Exemple

$$\begin{cases} S \rightarrow aTb \mid aU \\ T \rightarrow bTaTa \mid \varepsilon \\ U \rightarrow aU \mid b \end{cases} \Longrightarrow \begin{cases} S \rightarrow aTb \mid ab \mid aU \\ T \rightarrow bTaTa \mid baTa \mid bTaa \mid baa \\ U \rightarrow aU \mid b \end{cases}$$

Factorisation à gauche

- Lorsque l'on a:

  S → abcdeTb | abcdeUc
  T → bA
  U → cB
  …

- Il faut lire 6 caractères pour savoir quelle production appliquer

- L'idée est de factoriser

  S → abcde(Tb | Uc)

- S → abcdeS'
  S'→ Tb | Uc
  …

- Afin de différer la décision

Factorisation à gauche : Exemple

- $$\begin{cases} S \to aUbS \mid aUbSeT \mid a \\ T \to bcU \mid bca \\ U \to ba \end{cases}$$
- $$S \to a(\overbrace{UbS\ \underbrace{(\varepsilon \mid eT)}_{S'} \mid \varepsilon}^{S''})$$
- $$T \to bc\ (\overbrace{U \mid a}^{T'})$$
- $$\begin{cases} S \to aS'' \\ S'' \to UbS' \mid \varepsilon \\ S' \to eT \mid \varepsilon \\ T \to bcT' \\ T \to U \mid a \\ U \to ba \end{cases}$$