Grammaire ambigüe

Elimination de l'ambiguïté

- Ambiguïté pour les expressions de G_{arith} sur l'opérateur $\times$
  - Cause : G_{arith} accepte aussi bien l'associativité gauche que droite de $\times$
  - Solution : On choisit d'accepter ***l'associativité à gauche de $\times$***
    (e.g. l'interprétation $(1 \times 2) \times 3$ mais pas $1 \times (2 \times 3)$.)

- Ambiguïté pour les expressions de G_{arith} sur l'opérateur $+$
  - Cause : G_{arith} accepte aussi bien l'associativité gauche que droite de $+$
  - Solution : On choisit d'accepter ***l'associativité à gauche de $+$***
    (e.g. l'interprétation $(1 + 2) + 3$ mais pas $1 + (2 + 3)$.)

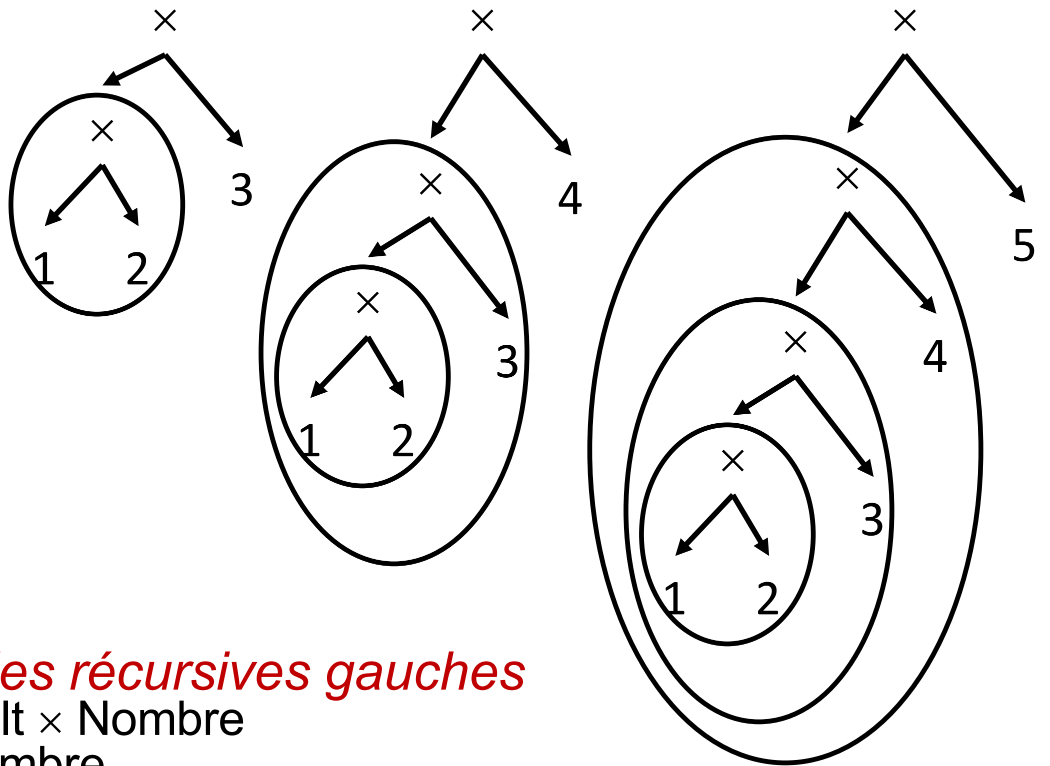
- Ambiguïté pour les expressions de G_{arith} sur les 2 opérateurs $\times$ *et* $+$
  - Cause : G_{arith} ne différencie pas entre les priorités de $\times$ et $+$
  - Solution : On choisit ***priorité($\times$) $>$ priorité($+$)***
    (e.g. l'interprétation $1 + (2 \times 3)$ mais pas $(1 + 2) \times 3$.)

- NB. Rien ne nous empêche d'adopter l'associativité droite !!
- Suite de l'exercice :
  - Essayer de réfléchir sur les transformations de G_{arith} pour supporter ces contraintes nécessaires à sa désambiguïsation

Grammaire ambigüe

Elimination de l'ambiguïté

- Accepter *l'associativité à gauche de ×*



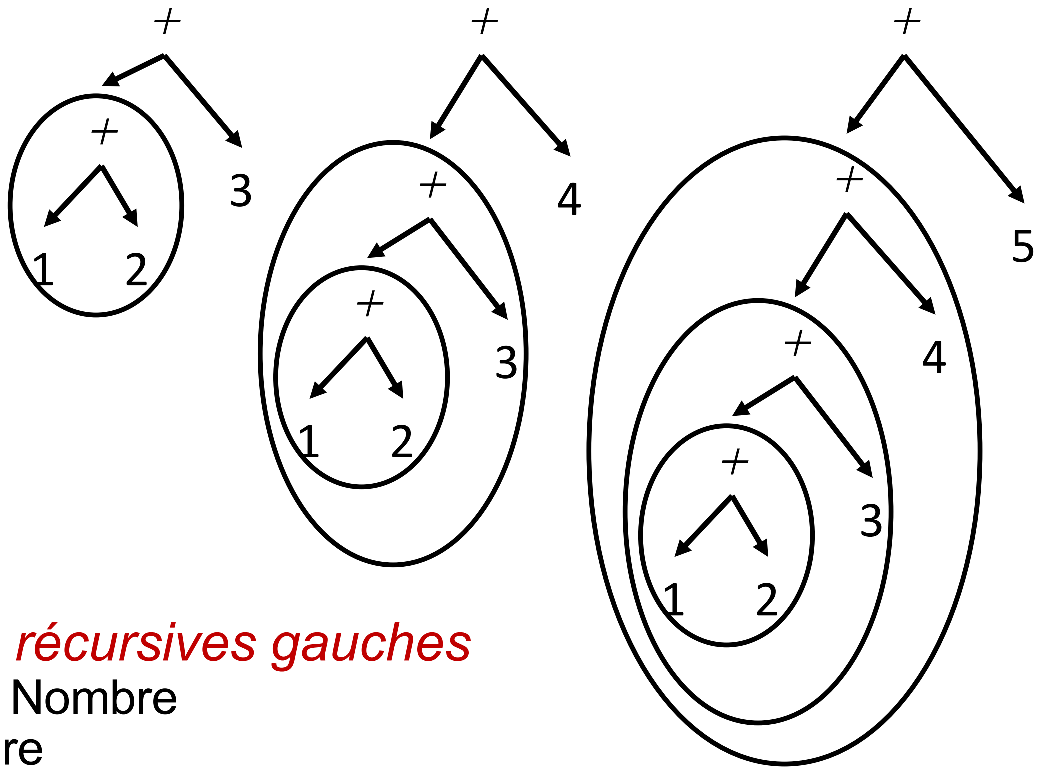
⇒ Introduire des *règles récursives gauches*

- R_{11} : Mult → Mult × Nombre
- R_{12} : Mult → Nombre

Grammaire ambigüe

Elimination de l'ambiguïté

- Accepter *l'associativité à gauche de +*



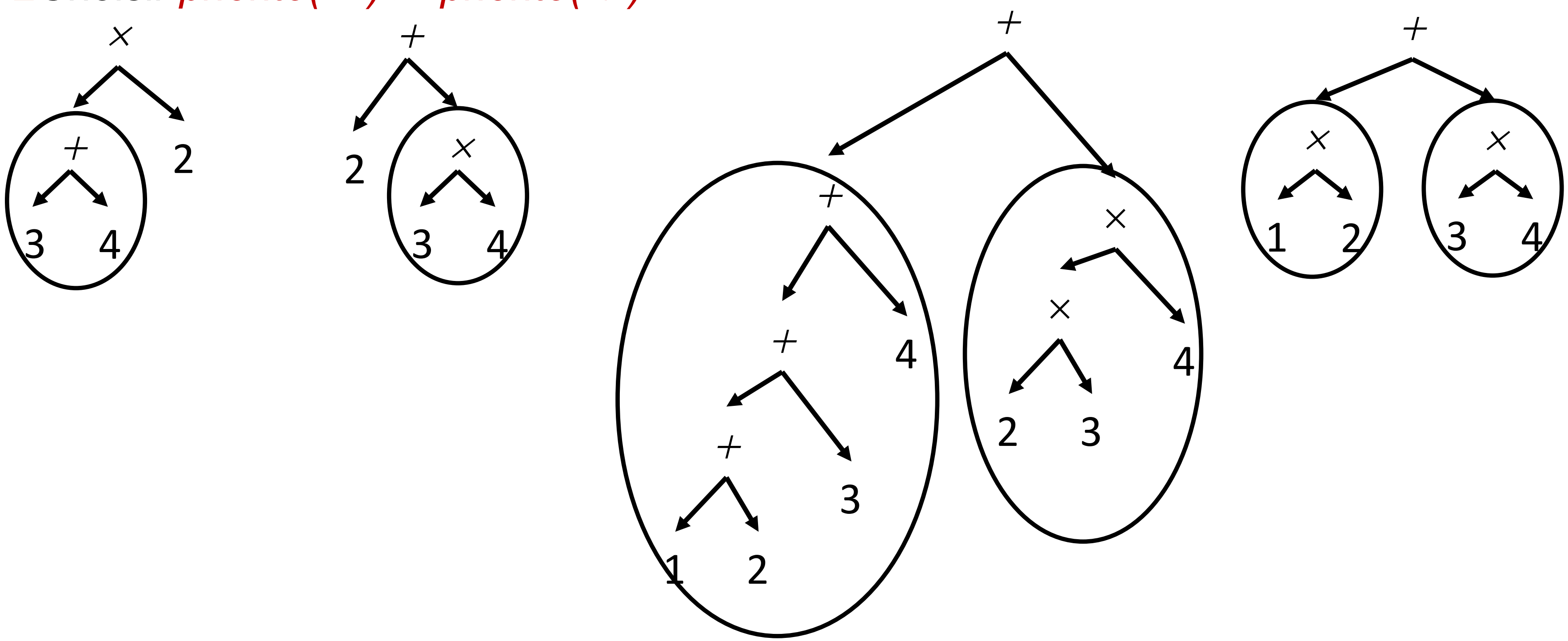
$\Rightarrow$ Introduire des *règles récursives gauches*

- R_{21} : Add $\rightarrow$ Add + Nombre
- R_{22} : Add $\rightarrow$ Nombre

Grammaire ambigüe

Elimination de l'ambiguïté

- Choisir *priorité(×) > priorité(+)*



⇒ Introduire des *règles récursives gauches*

- R_{31} : Add → Add + Mult (*plus générale que* R_{21}*, car* R_{12} : Mult → Nombre)
- R_{32} : Add → Mult (*plus générale que* R_{22}*, car* R_{12} : Mult → Nombre)

Grammaire ambigüe

Elimination de l'ambiguïté

- G'_{arith} < T={Nombre,+, ×}, NT ={Mult, Add}, S=Add, P = {R_{11}, R_{12}, R_{31}, R_{32}} >
  - R_{11} : Mult $\rightarrow$ Mult $\times$ Nombre
  - R_{12} : Mult $\rightarrow$ Nombre
  - ~~R_{21} : Add $\rightarrow$ Add + Nombre~~ (car *R_{31} est plus générale que R_{21}*)
  - ~~R_{22} : Add $\rightarrow$ Nombre~~ (car *R_{32} est plus générale que R_{22}*)
  - R_{31} : Add $\rightarrow$ Add + Mult
  - R_{32} : Add $\rightarrow$ Mult

- On a pris *S=Add* car *Add* est plus général de *Mult* (c.f. R_{32}, de *Add* on peut aller vers *Mult* mais l'inverse est faux)

  - G'_{arith} n'est pas ambiguë par construction !!

Grammaire ambigüe

Elimination de l’ambiguïté

- Défaut de G'_{arith}
  - C’est impossible de forcer
    - $1 + 2 \times 3$ à être interprété $(1 + 2) \times 3$
    - et $3 \times 1 + 2$ à être interprété $3 \times (1 + 2)$
- Suite de l’exercice :
  - Penser à une solution à ces nouveaux problèmes

Grammaire ambigüe

Elimination de l'ambiguïté

- Solution possible : introduire les parenthèses
  - NT devient {Mult, Add, Aux}
  - P devient $\{R'_{11}, R'_{12}, R'_{13}, R'_{14}, R_{31}, R_{32}\}$
    - R'_{11} : Mult $\rightarrow$ Mult $\times$ Aux *(plus générale que R_{11}, car* R'_{13} : Aux$\rightarrow$ Nombre*)*
    - R'_{12} : Mult $\rightarrow$ Aux *(plus générale que R_{12}, car* R'_{13} : Aux$\rightarrow$ Nombre*)*
    - R'_{13} : Aux $\rightarrow$ Nombre
    - R'_{14} : Aux $\rightarrow$ (Add)

    - Au lieu de
    - R_{11} : ~~Mult $\rightarrow$ Mult $\times$ Nombre~~
    - R_{12} : ~~Mult $\rightarrow$ Nombre~~

Grammaire ambigüe

Elimination de l'ambiguïté

- 4 cas avec deux opérations dans une expression parenthèsées
  - + à gauche du × : x + y + z
    - Cas particuliers : Nombre + Add, Add + Nombre
  - × à gauche du + : x + y × z
    - Exemples : (1 + 2) × 3
    - Cas particuliers : Nombre + Mult, Add × Nombre
  - + à gauche du + : x × y + z
    - Exemples : 3 × (1 + 2)
    - Cas particuliers : Nombre × Add, Mult + Nombre
  - × à gauche du × : x × y × z
    - Cas particuliers : Mult × Nombre, Nombre × Mult
- Seuls l'associativité gauche sera bien sûr prise en compte

Grammaire ambigüe

Elimination de l'ambiguïté

- G'_{arith} < T= {Nombre,+, ×}, NT = {Mult, Add, Aux}, S = Add, P = {R_1, R_2, R_3, R_4, R_5, R_6}>
  - R1 (R'_{11}) : Mult $\rightarrow$ Mult $\times$ Aux
  - R2 (R'_{12}) : Mult $\rightarrow$ Aux
  - R3 (R'_{13}) : Aux $\rightarrow$ Nombre
  - R4 (R'_{14}) : Aux $\rightarrow$ (Add)
  - R5 (R_{31}) : Add $\rightarrow$ Add + Mult
  - R6 (R_{32}) : Add $\rightarrow$ Mult

    - 1 + 2 × 3 pourra être interprété sans ambiguïté (1 + 2) × 3 :
      - Add $\Rightarrow_{R6}$ Mult $\Rightarrow_{R1}$ Mult $\times$ Aux $\Rightarrow_{R2}$ Aux $\times$ Aux $\Rightarrow_{R4}$ (Add) $\times$ Aux $\Rightarrow_{R5}$ (Add + Mult) $\times$ Aux $\Rightarrow_{R6}$ (Mult + Mult) $\times$ Aux $\Rightarrow_{R2}$ (Aux + Mult) $\times$ Aux $\Rightarrow_{R2}$ (Nombre + Mult) $\times$ Aux $\Rightarrow^*$ (Nombre + Nombre) $\times$ Nombre

    - 3 × 1 + 2 peut être interprété sans ambiguïté 3 × (1 + 2)
      - Add $\Rightarrow_{R6}$ Mult $\Rightarrow_{R1}$ Mult $\times$ Aux $\Rightarrow_{R2}$ Aux $\times$ Aux $\Rightarrow_{R3}$ Nombre $\times$ Aux $\Rightarrow_{R4}$ Nombre $\times$ (Add) $\Rightarrow^*$ Nombre $\times$ (Nombre + Nombre)

Analyseur Syntaxique
Types

- Il existe 2 types d'analyseurs syntaxiques (parseurs)
  - Analyseur descendant (top-down)
  - Analyseurs ascendants (bottom-up)

Analyseur Syntaxique
Types

- Analyseur descendant (top-down)
  - Algorithme :
    - Commence par la racine et procède en descendant l'arbre syntaxique jusqu'aux feuilles.
    - A chaque étape, l'analyseur choisit un nœud parmi les symboles non-terminaux et développe l'arbre à partir de ce nœud.
  - Exemples : Analyseur récursif descendant, Analyseur LL(1)

Analyseur Syntaxique

Types

- Analyseurs ascendants (bottom-up)
  - Algorithme :
    - Commence par les feuilles et procède en remontant l'arbre syntaxique jusqu'à la racine.
    - A chaque étape, l'analyseur ajoute des nœuds qui développent l'arbre partiellement construit.
  - Exemples : Analyseur LR(1), Analyseur LALR(1), Analyseur SLR(1)
    - $LR(1) \supset LALR(1) \supset SLR(1)$

Analyseur Syntaxique Descendant (top-down)

Analyseur Syntaxique Descendant Exemple

- Pour une grammaire du type
  - R1 : S $\rightarrow$ Mult
  - R2 : Mult $\rightarrow$ Mult $\times$ Aux
  - R3 : Mult $\rightarrow$ Aux
  - R4 : Aux $\rightarrow$ Nombre

- Problème :
  - L'algorithme risque de **développer** Mult en Mult $\times$ Aux puis en Mult $\times$ Mult $\times$ Aux puis en Mult $\times$ Aux $\times$ Aux $\times$ Aux … $\times$ Aux $\times$ Aux **indéfiniment** : **bouclage de l'analyseur** !!

- Solution :
  - **Éliminer la récursivité gauche** de la grammaire pour une analyse top-down

Analyseur Syntaxique Descendant
Récursivité a gauche

- Une règle de réécriture est dite récursive gauche si
  - Le premier symbole sur la partie droite de la règle est le même que celui sur sa partie gauche
    - Exemple :
      - $S \rightarrow Sa$
  - ou si le symbole de la partie gauche de la règle apparaît sur sa partie droite et tous les symboles qui le précèdent peuvent dériver le mot vide
    - Exemple :
      - $S \rightarrow TSa$
      - $T \rightarrow \varepsilon \mid \ldots$

Analyseur Syntaxique Descendant

- **G : Grammaire récursive gauche (NT = {S})**
  - $S \rightarrow S\ \alpha$ *(i.e. tout mot de L(G) se termine pas une suite de α)*
  - $\quad | \ \beta$ *(i.e. tout mot de L(G) commence par β ou contient seulement β)*
  - $S \Rightarrow S\ \alpha \Rightarrow S\ \alpha\ \alpha \Rightarrow \ldots \Rightarrow S\ \alpha^* \Rightarrow \underline{\beta\ \alpha^*}$
  - $\underline{L(G') = \beta\ \alpha^*}$

- **G' : Grammaire non-récursive gauche équivalente à G (NT' = {S,R})**
  - $S \rightarrow \beta\ R$ *(i.e. tout mot de L(G') commence par β)*
  - $R \rightarrow \alpha\ R$ *(i.e. tout mot de L(G') se termine éventuellement pas une suite de α)*
  - $\quad | \ \varepsilon$
  - $S \Rightarrow \beta\ R \Rightarrow \beta\ \alpha\ R \Rightarrow \alpha\ \alpha\ R \Rightarrow \ldots \Rightarrow \alpha^*\ R \Rightarrow \underline{\beta\ \alpha^*}$
  - $\underline{L(G') = \beta\ \alpha^*}$

- L(G) = L(G') donc la transformation de G en G' a conservé le langage reconnu

Elimination de la récursivité a gauche Solution

Entrée: Une grammaire G.

Sortie: Une grammaire équivalente sans récursivité à gauche.

Méthode:

Ordonner les non-terminaux par indices $A_1, ..., A_n$.

Pour i:=1 à n faire

Pour j : = 1 à i - 1 faire

Remplacer chaque production de la forme $A_i \rightarrow A_j\gamma$

par les productions $A_i \rightarrow \delta_1\gamma \mid ... \mid \delta_k\gamma$,

où $A_j \rightarrow \delta_1 \mid . . . \mid \delta_k$ sont les A_j-productions actuelles

Fin

Eliminer les récursivité immédiates à gauche des A_i-productions

Fin

Elimination de la récursivité a gauche Solution

- Éliminer la récursivité gauche de la grammaire suivante :
  - G'_{arith} < T={Nombre,+, ×}, NT={Mult, Add, Aux}, S=Add, P = {R_1, R_2, R_3, R_4, R_5, R_6}>
    - R1 : Mult $\rightarrow$ Mult $\times$ Aux
    - R2 : Mult $\rightarrow$ Aux
    - R3 : Aux $\rightarrow$ Nombre
    - R4 : Aux $\rightarrow$ (Add)
    - R5 : Add $\rightarrow$ Add + Mult
    - R6 : Add $\rightarrow$ Mult

Elimination de la récursivité a gauche Solution

Règles récursives gauche NT={Mult, Aux, Add}	Grammaire récursive droite équivalente (i.e. conserve l'associativité droite et les priorités de G'$_{arith}$) NT={Mult, Mult', Aux, Add, Add'}
	// **Transformation directe de Mult**
R_1 : Mult → Mult $\underline{\times}$ Aux	Mult → Aux Mult'
R_2 : Mult → Aux	
	Mult' → $\underline{\times}$ Aux Mult'
	| ε
	// **Aux n'est pas récursive gauche**
R_3 : Aux → <u>Nombre</u>	Aux → <u>Nombre</u>
R_4 : Aux → <u>(</u> Add <u>)</u>	Aux → <u>(</u> Add <u>)</u>
	// **Transformation directe de Add**
R_5 : Add → Add $\underline{+}$ Mult	Add → Mult Add'
R_6 : Add → Mult	Add' → $\underline{+}$ Mult Add'
	|ε

Elimination de la récursivité a gauche Solution

- Éliminer la récursivité gauche de la grammaire suivante :

- $S \rightarrow S + A \mid S \times B \mid C \mid D$

Elimination de la récursivité a gauche Solution

$\boldsymbol{S} \rightarrow \boldsymbol{S}\,\alpha_1 \mid \boldsymbol{S}\,\alpha_2 \mid \boldsymbol{S}\,\alpha_3 \mid \ldots \mid \boldsymbol{S}\,\alpha_n \mid \beta_1 \mid \beta_2 \mid \ldots\ldots \mid \beta_n$ ➔ $\boldsymbol{S} \rightarrow \beta_1\,\boldsymbol{R} \mid \beta_2\,\boldsymbol{R} \mid \ldots\ldots \mid \beta_n\,\boldsymbol{R}$

$\boldsymbol{R} \rightarrow \alpha_1\,\boldsymbol{R} \mid \alpha_2\,\boldsymbol{R} \mid \ldots \mid \alpha_n\,\boldsymbol{R} \mid \varepsilon$

- $S \rightarrow S + A \mid S\, x\, B \mid C \mid D$

- $S \rightarrow C\, S' \mid D\, S'$
- $S' \rightarrow + A\, S' \mid x\, B\, S' \mid \varepsilon$

Factorisation à gauche
Algorithme

- *Pour chaque $A \in NT$*
  - *Trouver le plus long préfixe « α » commun à deux ou plusieurs parties droites de règles relatives à A*
  - *Si ($\alpha \neq \varepsilon$) alors*
    - *remplacer les règles de A de la forme*
      - $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \cdots \mid \alpha\beta_n \mid \gamma$
    - *par :*
      - $A \rightarrow \alpha\, B \mid \gamma$
      - $B \rightarrow \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$
      - $NT \leftarrow NT \cup \{B\}$ *// ajouter B à l'ensemble des symboles non-terminaux*
- *Répéter jusqu'à ce qu'il n'y ait plus de préfixe commun*

Factorisation à gauche
Exemple

- $S \rightarrow aS \mid aB \mid aC \mid aD$

Simple factorisation

- $S \rightarrow a\, S'$
- $S' \rightarrow S \mid B \mid C \mid D$

Factorisation à gauche

Exercice

Factorisation à gauche de:

- $S \rightarrow$ *aA
- |*aB
- |*C

Factorisation à gauche
Solution

- $S \rightarrow$ *aA | *aB | *C

- $S \rightarrow$ *aS' | *C
- $S' \rightarrow$ A | B

- $S \rightarrow$ *S''
- $S'' \rightarrow$ aS'' | C

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)

LL : Left to right – Leftmost derivation

- Commence par la racine et procède en descendant l'arbre syntaxique jusqu'aux feuilles.

- A chaque étape, l'analyseur choisit un nœud parmi les symboles non-terminaux et développe l'arbre à partir de ce nœud.

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)
LL : Left to right – Leftmost derivation

- Une grammaire est dite **LL(k)** si, et seulement si, elle peut être analysée en ne disposant, à chaque instant, que des **n** prochains **terminaux** non encore consommés.

- Le **k** appelé lookahead (regarder en avant) indique le nombre de **terminaux** qu'il faut avoir lus sans les avoir encore consommés pour décider quelle dérivation faire.

- PL'analyse d'une grammaire **LL(3)** impose de gérer 3 variables contenant les 3 prochains **terminaux** non encore consommés à chaque instant.

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)

LL : Left to right – Leftmost derivation

- Soit $\boldsymbol{A \rightarrow \alpha}$ une règle de production, on définit **First(α)** =
  - {ensemble des symboles terminaux qui peuvent apparaître comme premiers symboles dans les mots dérivables à partir de α}

- Soit ***G* = <*T*, *NT*, *S*, *P*>** une grammaire, G est dite prédictive (dite LL(1)) ssi
- $\forall$ les règles $A \rightarrow \alpha_1 \mid \alpha_2 \mid \alpha_3 \mid \cdots \alpha_n$ de **P**
  - $\forall$ i,j $First(\alpha_i) \cap First(\alpha_j) = \varnothing$

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)

Exemple

- La grammaire S → a | bS est **LL(1)**:
  - Productions(S→a) = First(a) = {a}
  - Productions(S→ bS) = First(bS) = First (b) = {b}

- La grammaire S → a | aS n'est pas **LL(1)**:
  - Productions(S→a) = First (a) = {a}
  - Productions(S→ aS) = First (aS) = First (a) = {a}

Analyseur Syntaxique Descendant
Grammaire prédictive LL(1)
First/Follow

- L'ensemble premier :

Productions(S→a) = **Premier**(a) si a n'est pas vide.

- Si a est de la forme aS ou a est un **terminal** alors nous avons **First**(aS) = {a}.

- Si a est de la forme AS ou A est un **non-terminal** alors nous avons **First**(AS) = **First**(A).

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)

First/Follow

- L'ensemble suivant :

Productions(S→a) = **Follow**(S) si a est vide.

- Si une règle est de la forme S → aTb alors l'ensemble **Follow**(T) contient l'ensemble **First**(b).
- Si une règle est de la forme A→ aT alors l'ensemble **Follow**(T) contient l'ensemble **Follow**(A).

Analyseur Syntaxique Descendant
Grammaire prédictive LL(1)
Exercice : First

Calculer l'ensemble First (S) des règles suivantes :

- $S \rightarrow Aa$
- $A \rightarrow bB$
- $\quad\quad | \ \varepsilon$

Calculer l'ensemble First des non terminaux de la grammaire suivante :

- $S \rightarrow ETC$
- $E \rightarrow aE \mid \varepsilon$
- $T \rightarrow bT \mid cT \mid \varepsilon$
- $C \rightarrow dC \mid da \mid dE$

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)

Solution: First

- $S \rightarrow Aa$
- $A \rightarrow bB$
- $\quad\quad | \ \varepsilon$

First (S) = First (A) + First(S si A=ε) = {b,a}

- $S \rightarrow ETC$
- $E \rightarrow aE \mid \varepsilon$
- $T \rightarrow bT \mid cT \mid \varepsilon$
- $C \rightarrow dC \mid da \mid dE$

First (S) = {a,b,c,d}

First (E) = {a,ε}

First(T) = {b,c,ε}

First(C) = {d}

Analyseur Syntaxique Descendant
Grammaire prédictive LL(1)
Exercice : Follow

Calculer l'ensemble Follow des non terminaux de la grammaire suivante :

- $S \rightarrow ETC$
- $E \rightarrow aE \mid \varepsilon$
- $T \rightarrow bT \mid cT \mid \varepsilon$
- $C \rightarrow dC \mid da \mid dE$

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)
Solution: Follow

- $S \rightarrow ETC$
- $E \rightarrow aE \mid \varepsilon$
- $T \rightarrow bT \mid cT \mid \varepsilon$
- $C \rightarrow dC \mid da \mid dE$

First (S) = {a,b,c,d}

First (E) = {a,ε}

First(T) = {b,c,ε}

First(C) = {d}

Follow (S) = {$}

Folow (E) = First(T)+First(C)={b,c,d,$}

Follow(T) = First(C) = {d}

Follow (C) = Follow(S) = {$}

Analyseur Syntaxique Descendant
Grammaire prédictive LL(1)
Exemple

- Déterminons un programme permettant une analyse syntaxique en reprenant la grammaire G suivantes:
  - expr ➔ term
  - | term “+“ expr
  - | term “-“ expr
  - term ➔ fact
  - | fact “*“ term
  - | fact “/“ term
  - fact ➔ “a“ | “b“ | “c“ | “d“
  - | (“expr“)
  - | fact “^“fact
  - expr /*axiome*/)
- Nous décrirons seulement les règles de productions de fact.

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)
Exemple

- fact ➔ “a“ | “b“ | “c“ | “d“
- | (“expr“)
- | fact “^“fact

Pour la fonction fact, nous obtenons alors :

```
void _fact()
{
 if (sTerminal == “ ( “) {  /*commençons par “(“ expr “)“ */
 _expr();
 if (sTerminal != “ ) “)
      {_ErrSyntax("après expr “ ) “ attendu");}
 else
      {_avancer();}
 } /*fin de “(“ expr “)“ */
else
 {
```

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)

Exemple

- fact ➔ “a“ | “b“ | “c“ | “d“
- | (“expr“)
- | fact “^“fact

```
{
  if (sTerminal == (“a“ || “b“ || “c“ || “d“) {
       _avancer()
   }
  else
   {
     _fact();
     if (sTerminal != “^“)
          {_ErrSyntax(‘’après FACT “^“ attendu ’’);}
     else {
        _fact();
        _avancer();
        } /*fin de fact “^“ fact*/
    }
} /*Fin de la fonction pour les règles de fact*/
```

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)

- Nous venons de produire un programme permettant de construire uniquement une analyse de la production Fact. Mais pouvons-nous être plus général pour l'ensemble des constructions ?

- L'**analyse prédictive** se fait à l'aide d'une **table d'analyse syntaxique prédictive**.

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)
Table d'Analyse

First (S) = {a,b,c,d}

First (E) = {a,ε}

First(T) = {b,c,ε}

First(C) = {d}

Follow (S) = {$}

Folow (E) = First(T)+First(C)={b,c,d,$}

Follow(T) = First(T) = {d}

Follow (C) = Follow(S) = {$}

	First	Follow
S	{a,b,c,d}	{$}
E	{a,ε}	{b,c,d,$}
T	{b,c,ε}	{d}
C	{d}	{$}

Analyseur Syntaxique Descendant
Grammaire prédictive LL(1)
Table d'Analyse

- $S \rightarrow ETC$
- $E \rightarrow aE \mid \varepsilon$
- $T \rightarrow bT \mid cT \mid \varepsilon$
- $C \rightarrow dC \mid da \mid dE$

$C \rightarrow dC'$

$C' \rightarrow C \mid a \mid E$

	First	Follow
S	{a,b,c,d}	{$}
E	{a,ε}	{b,c,d,$}
T	{b,c,ε}	{d}
C	{d}	{$}

	a	**b**	**c**	**d**	**$**
S	$S \rightarrow ETC$	$S \rightarrow ETC$	$S \rightarrow ETC$	$S \rightarrow ETC$	
E	$E \rightarrow aE$	$E \rightarrow \varepsilon$	$E \rightarrow \varepsilon$	$E \rightarrow \varepsilon$	$E \rightarrow \varepsilon$
T		$T \rightarrow bT$	$T \rightarrow cT$	$T \rightarrow \varepsilon$	
C				$C \rightarrow dC$ $C \rightarrow da$ $C \rightarrow dE$	

Table d'Analyse

N'est pas une grammaire LL(1)

Factorisation

Analyseur Syntaxique Descendant
Grammaire prédictive LL(1)
Exercice

Soit la grammaire suivante :

- $S \rightarrow S + T \mid T$
- $T \rightarrow T*F \mid F$
- $F \rightarrow (S) \mid id$

Construire la table d'analyse

Analyseur Syntaxique Descendant
Grammaire prédictive LL(1)
Solution

Etape 1: Supprimer la récursivité a gauche

- $S \to S + T \mid T$
- $T \to T*F \mid F$
- $F \to (S) \mid id$

- $S \to TS'$
- $S' \to + T S' \mid \varepsilon$
- $T \to FT'$
- $T' \to *FT' \mid \varepsilon$
- $F \to (S) \mid id$

Nouvelle grammaire

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)

Solution

Calculer l'ensemble First et Follow

- $S \rightarrow TS'$
- $S' \rightarrow + T S' \mid \varepsilon$
- $T \rightarrow FT'$
- $T' \rightarrow *FT' \mid \varepsilon$
- $F \rightarrow (S) \mid id$

	First	Follow
S	{ (, id }	{) , $ }
S'	{ + , ε}	{) , $ }
T	{ (, id }	{ + ,) , $ }
T'	{ * , ε}	{ + ,) , $ }
F	{ (, id }	{ * , + ,) , $ }

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)
Solution

- $S \rightarrow TS'$
- $S' \rightarrow + T S' \mid \varepsilon$
- $T \rightarrow FT'$
- $T' \rightarrow *FT' \mid \varepsilon$
- $F \rightarrow (S) \mid id$

	First	Follow
S	{ (, id }	{) , $ }
S'	{ + , ε}	{) , $ }
T	{ (, id }	{ + ,) , $ }
T'	{ * , ε}	{ + ,) , $ }
F	{ (, id }	{ * , + ,) , $ }

Table d'analyse

	id	**+**	*****	**(**	**)**	**$**
S	$S \rightarrow TS'$			$S \rightarrow TS'$		
S'		$S' \rightarrow + T S'$			$S' \rightarrow \varepsilon$	$S' \rightarrow \varepsilon$
T	$T \rightarrow FT'$			$T \rightarrow FT'$		
T'		$T' \rightarrow \varepsilon$	$T' \rightarrow *FT'$		$T' \rightarrow \varepsilon$	$T' \rightarrow \varepsilon$
F	$F \rightarrow id$			$F \rightarrow (S)$		

Analyseur Syntaxique Descendant

Grammaire prédictive LL(1)

Exemple: Vérification de mot

- Prenons la grammaire G suivante :

  - S → Term Exp
  - Exp → +Term Exp | ε
  - Term → Entier ou “0“ | “1“ | .. | “9“ | ..

Analyseur Syntaxique Descendant
Grammaire prédictive LL(1)
Exemple: Vérification de mot

- S → Term Exp
- Exp → +Term Exp | ε
- Term → Entier ou "0" | "1" | .. | "9" | ..

	First	Follow
S	{ entier }	{ $ }
Exp	{+ , ε}	{ $ }
Term	{ entier }	{+ , $ }

Table d'analyse

	+	**entier**	**$**
S		S → Term Exp	
Exp	Exp → +Term Exp		Exp → ε
Term		Term → Entier	

Analyseur Syntaxique Descendant
Grammaire prédictive LL(1)
Exemple: Vérification de mot

Le mot :1+3

Reconnu	Pile	Entrée	Action
	S$	1 + 3$	S → Term Exp
	Term Exp$	1 + 3$	Term → Entier
	Entier Expr$	1 + 3$	Entier = 1 (dépilé Entier)
1	Expr$	+ 3$	Exp → +Term Exp
1	+Term Exp$	+ 3$	+=+ (Dépilé +)
1+	Term Exp$	3$	Term → Entier
1+	Entier Expr$	3$	Entier = 3 (dépilé Entier)
1+3	Expr$	$	Exp → ε
	$	$	Accepté